STACK DATA ENGINE

Description and Implementation

by

G. Efland

R.C. Mosteller

Technical Report #3364

December 1979

Computer Science Department

California Institute of Technology

Pasadena, California 91125

Silicon Structures Project

sponsored by

Burroughs Corporation, Digital Equipment Corporation,

Hewlett Packard Company, Honeywell Incorporated,

International Business Machines Corporation,

Intel Corporation, Xerox Corporation,

and the National Science Foundation

12/1/79

G R

STACK DATA ENGINE

Description and Implementation

Greg Efland

R. C. Mosteller

December 79

TABLE OF CONTENTS

A P P E N D I C E S

1. INTRODUCTION

This report is a description of the design and implementation of the GR microprocessor, an eight bit stack machine with hierachical procedure calls. The processor is an experiment in computer implementation, and features a microcoded control section. Interface to memory and I/O is provided in order to test operation of the machine.

The total time from start to completion of this project was two and one half weeks, not including an extra week for completion of this document. As a result of this schedule certain shortcuts were taken, particularly in the layout of the chip. The experience gained has been invaluable, though, and we hope to complete a successor to this version in the upcoming year. The results of many of the mistakes have been summarized in section 5, where they can hopefully benefit others.

This chip was designed partially as a project for CS-181 class at Caltech. The architecture and instruction set are based on earlier work done on a Pascal compiler for CS-138 class at Caltech during the winter of 1979.

The GR acronym stands, of course, for Greg and Richard. We would like to thank Dave Johannson for supplying a data path to our specification via Bristle Blocks, and his general counseling on the design.

2. MACHINE DESCRIPTION

2.1 Macro Architectural

2.1.1 Programmer's View

The GR machine contains five registers that are used for program and data manipulation. They are defined as follows:

PC - This is the program counter register. It points to the current operator in the code memory area. This register is automatically maintained by the processor.

BASE - This points to the currently activated procedure in the program stack. All references from the load, store, and procedure return instructions are relative to the BASE register. This register is automatically maintained by the processor.

TOP - This points to the top of the data stack. This register is automatically maintained by the processor.

A - This is the top register in the stack. All stores and loads are done with this register. All unary operations are performed on this register while binary operation are performed on the A and B registers.

B - This register is the next to top register in the stack. It is used as an operand and result register for the binary operators.

The number base for the GR engine is two's complement. This allows addressing arithmetic to use negative displacements for accessing parameters. However, all code addresses represent absolute locations.

2.1.2 Stack

The data stack provides the program running environment and data storage area for the GR engine. The processor is linked to the stack by the BASE and TOP registers as defined above. Generally there is no direct access to the BASE, TOP, A, and B registers. They are maintained dynamically by the processor.

The procedural environment is defined by the word pointed to by BASE and the two words below it, as follows:

BASE)- Points to the procedure activation record of the next higher textual level. This is a chain up the stack in lexicographical order of procedure calls. Procedure variables may be accessed through this chain at any level (see the LOD and STO operators). It is important to note that this word is not automatically set up by the CALL instruction. It should be initialized by a LOD with the OFFSET field equal to 0 and the difference field equal to the addressing level desired.

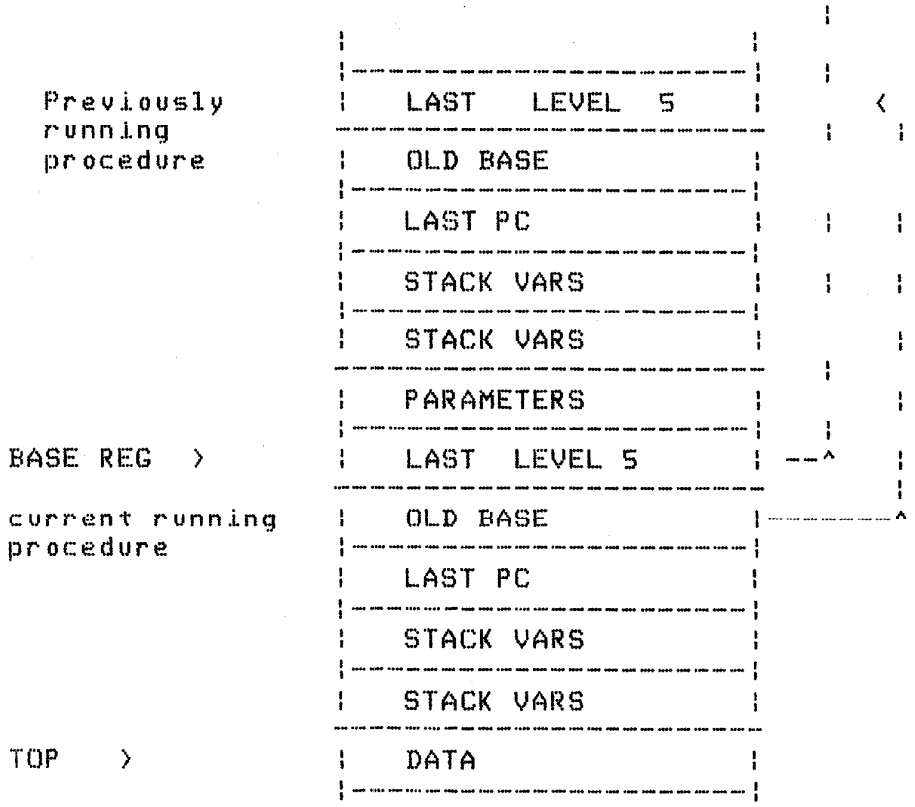
BASE+1)- This word points to the prior base address from which this procedure was called. That is, each procedure call is chained through the stack. The CALL operator is used to initialize this.

BASE+2)- This is the address of the next operator to be executed after the return from the procedure call. Initialized via the CALL instruction.

BASE-M)- This area is used for procedural parameter passing.

BASE+2+n)- This area is used for storage of variables local to this procedure.

S T A C K E X A M P L E



2.1.3 Instruction Set

The instruction set of the GR microprocessor is divided into six classes: arithmetic, data manipulation, subroutine handling, comparison, branching, and stack manipulation. Each of the instructions is defined below, with a summary provided in figure 2.1.3A.

Arithmetic

The arithmetic operators use the A and B registers as their arguments. The result is put into the B register.

ADD - adds the top two numbers in the stack.

SUB - subtracts the top two numbers in the stack.

OR - logically or's the top two numbers in the stack.

AND - logically and's the top two numbers in the stack.

NOT - logically compliments the top word in the stack.

Data Manipulation

The data manipulation operators provide for bringing variables to and from the top of stack. The A register represents the top of stack, while variables are accessed at an effective address that is determined from the next two syllables in the instruction. The first syllable, known as the textual level difference or 'dif' field, defines which procedure activation record contains the variable being accessed. The second syllable, the 'off' field, is the offset from the base of that activation record to where the variable is stored. The following algorithm is used to calculate the effective address (or ea) of the variable:

```
ea <- BASE
n <- dif
while (n>0) do
     ea <- mem[ea]
     n <- n-1
ea <- ea+off
```

LOD - This operator loads the top of the stack from the address in the data area defined by the 'dif' and 'off' instruction fields.

STO - This operator stores the top of the stack at the address in the data area defined by the the 'dif' and 'off' instruction fields.

LIT - This operator places the next syllable of the instruction onto the top of stack.

Branching

The branching operator allows control flow to be altered as the result of data calculations.

JCT - Conditionally branches to the code address contained in the instruction, based on the top of stack. If the lsb of the top of stack is set, the branch is taken. Otherwise the next instruction is executed.

Subroutine handling

The subroutine operators provide the mechanism for activating and exiting procedures, and in particular allow for initialization and removal of the procedure activation record described in the stack section above (2.1.2). Note that in practice a combination of instructions would be used when calling and returning from a procedure.

```
eg.  to call a proc.    PUSH          ! save A reg
                        LOD  n+1,0    ! link textual level
                        PUSH          ! save in stack
                        PUSH
                        CALL addr
                        .
                        .

     proc.      addr:   PUSH          ! save in stack
                        PUSH          ! procedure body
                        .

     to exit the proc.  POP           ! restore saved data
                        POP
                        EXIT          ! go back
```

CALL - This operator calls the procedure whose address is in the next syllable of the instruction.

EXIT - This operator exits from a procedure.

Comparison operator

The comparison operator provides for testing the results of previous operations and setting a flag suitable for use by the conditional branch instruction.

EQ - If the top two items in the stack are equal all ones are put on the top of stack (representing TRUE). Otherwise the top of stack is cleared (left FALSE).

Stack Manipulation

These operators are used to prevent overflow and underflow of the stack from the A and B registers.

PUSH - This operator increments the TOP register by one, stores the B register at the address in TOP, and then puts the A register into the B register.

POP - This operator puts the contents of the B register into the A register. It then reads from the memory location addressed by TOP the new value of the B register. Finally the TOP register is decremented by one.

INSTRUCTION SET SUMMARY

	⟨OPCODE⟩	⟨FIELDS⟩	⟨OPERATION⟩
ADD	⟨1⟩		B ⟨- B+A
SUB	⟨2⟩		B ⟨- B-A
AND	⟨3⟩		B ⟨- B and A
OR	⟨4⟩		B ⟨- B or A
NOT	⟨5⟩		A ⟨- not A
LIT	⟨6⟩	⟨n⟩	A ⟨- n
LOD	⟨7⟩	⟨dif⟩⟨off⟩	A ⟨- mem[ea]
STO	⟨8⟩	⟨dif⟩⟨off⟩	mem[ea] ⟨- A
CALL	⟨9⟩	⟨addr⟩	B ⟨- BASE; A ⟨- PC; BASE ⟨- TOP; PC ⟨- addr
EXIT	⟨10⟩		TOP ⟨- BASE-1; BASE ⟨- B; PC ⟨- A
EQ	⟨11⟩		B ⟨- B=A
JCT	⟨12⟩	⟨addr⟩	if lsb(A)=1 then PC ⟨- addr
PUSH	⟨13⟩		TOP ⟨- TOP+1; mem[TOP] ⟨- B; B ⟨- A
POP	⟨14⟩		A ⟨- B; B ⟨- mem[TOP]; TOP ⟨- TOP-1

NOTES:
1. TOP points to the top of stack in memory.
2. mem[reg] represents the memory location pointed to by register reg.
3. All opcodes and fields are 8 bits.
4. ⟨dif⟩ and ⟨off⟩ fields are two's complement.
5. 'ea' is the effective address formed by following the pointer chain from BASE up ⟨dif⟩ levels, and then adding ⟨off⟩ to that pointer.

FIGURE 2.1.3A

2.2 Micro Architecture

An internal view of the GR machine can be found in figure 2.2.1A. This represents the view of the processor seen by the microprogrammer.

The microcode format and field descriptions can be seen in figure 2.2.2A, while the actual values of the field constants are located in figure 2.2.2B. A description of the bus control machine is in figure 2.2.3A.

2.2.1 Internal Bus Structure

The data path section of the machine is centered around two busses, known as B1 and B2 (see fig. 2.2.1A). B2 is the main bus in that the register file can be read or written from it, and it is the only access path to the PORT. B1 is only used for reading the register file.

A microinstruction cycle starts at the beginning of PH1. At this time the selected register is read onto B1 and latched into one operand register of the ALU. At the same time a transfer is made on B2. This is either a read or write to the selected register according to the microcode. In either case the data on B2 is latched into the second operand register of the ALU. In the event that a register is being written from B2, the source of data can be either the ALU (result of previous calculation) or the PORT (containing data from the outside).

The PH2 clock period is reserved for the ALU operation. The selected function is applied to the two operand registers and the result saved in the output register where it can be used in the next PH1 transfer.

The microcode sequencer runs in parallel with the data path. During PH1 the generation of the next microcode address is accomplished. PH2 is then used to read the contents of the ROM location. Of course, lines such as the ALU function controls are sampled during PH1 to keep things working properly.

The operation of the processor's external bus interface is not controlled directly by the microcode but rather via the bus controller machine. This finite state machine receives instructions from the microcode to execute various types of bus cycles, which it does while the microcode is executing other tasks. The machine controls the operation of the port latching and tristate enable signals, as well as the external bus interface signals.

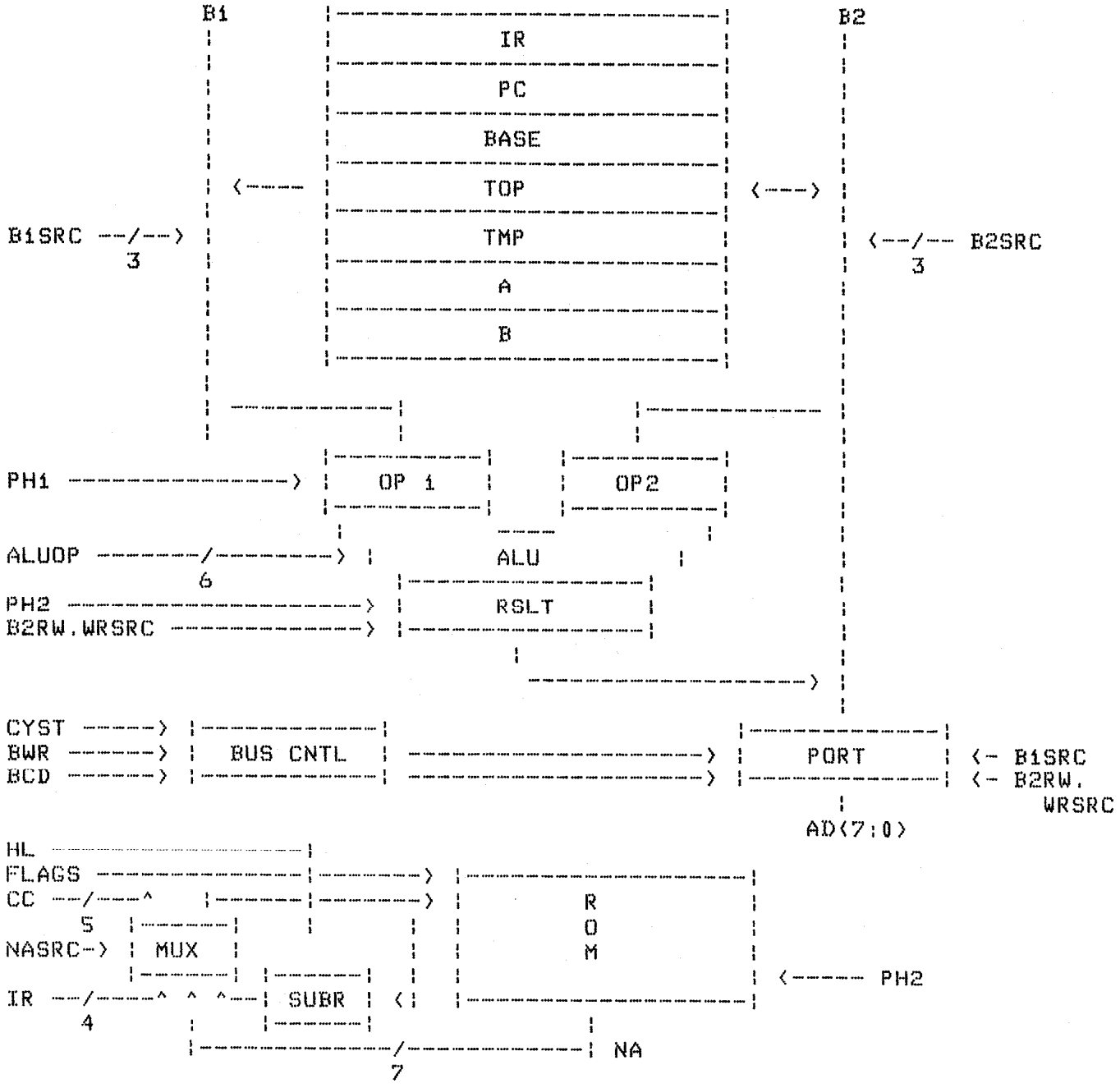

B1
B2
IR
PC
BASE
TOP
TMP
A
B
B1SRC
3
B2SRC
3
PH1
OP 1
OP2
ALUOP
6
ALU
PH2
B2RW.WRSRC
RSLT
CYST
BWR
BCD
BUS CNTL
PORT
B1SRC
B2RW.
WRSRC
AD<7:0>
HL
FLAGS
CC
5
NASRC
MUX
R
O
M
PH2
IR
4
SUBR
NA
7

FIGURE 2.2.1A

2.2.2 Ucode Fields

This section contains a description of the fields making up the microcode, figure 2.2.2A below. The field values are in figure 2.2.2B below.

UCODE	FIELD	DESCRIPTION
CONTROL:	NA(6:0)	Next Address field. Used as the address of the next microinstruction when selected by the NASRC field.
	NASRC(2:0)	Selects the Next Address Source - either the IR reg., Subroutine reg., or the NA field.
	CC(4:0)	Selects the low order bit of the next microinstruction address. Either the ALU LSB, MSB, or zero flag, the RDY flag, or the NA field LSB is selected.
	HL	Hold or Latch the current address in the Subroutine reg.
OPERATION:	ALUOP(5:0)	Select ALU function. Add, subtract, increment, decrement, and, or, not, pass, zero, ones supported.
OPERANDS:	B1SRC(2:0)	Specifies register whose contents are to be placed on B1.
	B2SRC(2:0)	Specifies the register whose contents are to be placed on B2 or written from B2.
	B2RW	Specifies whether the register named in B2SRC is to be read or written.
	WRSRC	When B2RW specifies a write, this specifies either the ALU output or the PORT as the source of data for B2.
BUS:	CYST	Specifies that a memory access is to be begun to the address currently being latched into the PORT.
	BRW	Specifies whether the external bus cycle being started is a read or write cycle.
	BCD	Specifies whether the external bus cycle being started references code or data memory.

FIGURE 2.2.2A

FIELD VALUES

NASRC	
001	Instruction Register
010	Subroutine Register
100	NA Field

CC	
00001	Ready Flag
00010	ALU MSB
00100	ALU Zero
01000	ALU LSB
10000	NA Field LSB

HL	
0	Hold
1	Latch

ALUOP	
000011	Decrement B2
000110	Add B1, B2
10000	Zero
010011	Not B2
011000	And B1, B2
011010	Pass B1
011110	Or B1, B2
011111	Ones
101001	Sub B1, B2 (B2-B1)
101100	Increment B2

B1SRC	
000	A
001	B
010	BASE
011	TOP
100	PC
101	IR
110	TMP
111	PORT (write from B2)

B2SRC	
000	A
001	B
010	BASE
011	TOP
100	PC
101	IR
110	TMP

B2RW	
0	Read
	Write

WRSRC	
0	ALU
1	PORT

CYST	
0	Idle
1	Start bus cycle

BRW	
0	Read Cycle
1	Write Cycle

BCD	
0	Code Space Access
1	Data Space Access

FIGURE 2.2.2B

2.2.3 Bus controller

The bus controller receives commands from the microcode, as described above, to run external cycles over the memory-I/O bus. The controller generates the external bus control signals (AS, DS, C/D, R/W) as well as the output enable and input latch controls for the PORT. Implemented as a finite state machine, the logic equations for the controller are given below. The PLA code can be found in Appendix F.

```
S0   =   [ _S0 _S1 MST + _S0 S1 _RDY ] _RES

S1   =   [ S0 _S1 + _S0 S1 + S0 S1 _RDY] _RES

AS   =   _S0 _S1  MST

DS   =   S0 S1 _RDY + _S0 S1

C/D  =   MST BCD + _MST R/W

R/W  =   MST _BRW + _ST R/W

EN   =   _S0 S1 _R/W + S0 S1 _RDY _R/W

LTCH =   _S0 S1 + S0 S1 R/W _RDY
```

FIGURE 2.2.3A

2.3 EXTERNAL INTERFACE

This section describes interfacing the GR microprocessor to the outside world. A summary of the pin connections can be found in figure 2.3A, with a more in-depth description below.

2.3.1 Memory-I/O Bus

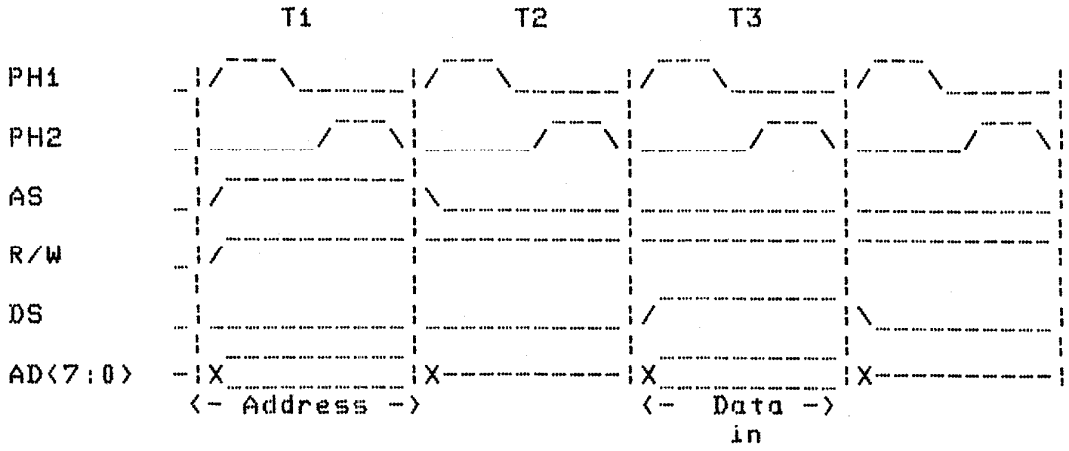
The GR microprocessor is intended to interface to standard memory chips and I/O devices via its bus interface. This consists of an 8 bit bidirectional address/data bus and five control lines. A timing diagram of a typical read and write cycle on the bus can be seen in figure 2.3.1A, which is useful for the explanation of the control signals below.

The start of an external bus cycle is indicated by the AS (address strobe) signal. This indicates that a valid address is on the bus lines and should be latched into the external memory address register. At the same time, the R/W and C/D outputs become valid. These signals indicate whether a read or write cycle is begining, and whether code or data memory is being accessed. They remain active for the duration of the bus cycle.

After one idle clock period the DS (data strobe) signal will be asserted. In the case of a read cycle this indicates that data from the external source should be placed on the bus lines. In the case of a write cycle it indicates data from the processor is available on the bus.

At the end of any clock cycle in which DS is asserted the RDY input will be sampled. If it is low, the processor will suspend operation in the next cycle and wait for the external memory or I/O to respond. All control lines (including DS) will remain active for as many idle cycles as necessary. The processor will resume operation at the start of the first cycle after the RDY input goes high. Note that the RDY input is level sensitive, and thus may be permanently tied high if the minimum 3 clock bus cycle can be met by memory and devices attached to the bus.

READ CYCLE



WRITE CYCLE

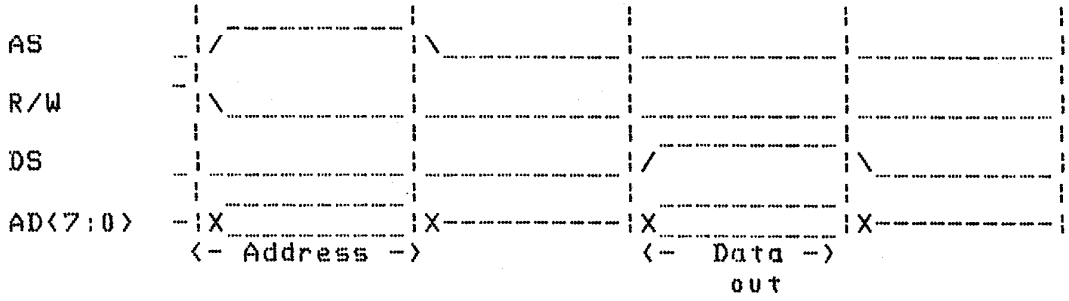


FIGURE 2.3.1A

2.3.2 External Inputs

In addition to the bus interface lines, three other inputs must be provided for proper operation of the GR microprocessor.

The two clock inputs (PH1 and PH2) should be non-overlapping and swing from 0 to VDD (no on-chip buffering). PH2 includes the ALU operation and ROM discharge while PH1 is for next microaddress generation. One would thus expect PH2 to be the longer of the two. (We are not too optimistic about the speed of version 0 of the machine).

The RESET input is used to initialize the processor, as for example after power-up. It clears the internal registers and provides a known address for initial code execution. The processor will begin running at the start of the first clock cycle after the RESET input is negated. However, instruction execution will not start for four clock periods while the initialization microcode runs.

P I N S

VD - Power supply input.

GND - Ground connection.

VBIAS - Negative bias supply input.

PH1,PH2 - Clock inputs.

RESET - Initializes the processor to the known condition:

```
PC   <- 0
BASE <- 0
TOP  <- 0
```

The same as executing instruction 00H. This input should be synchronized to the leading edge of phase 2.

AD<7:0> - Tristate address/data bus. AD0 is the least significant bit.

AS - Address Strobe. Indicates the start of a memory cycle, and that a valid address is on the AD pins. The address is stable and should be latched during (phase 2 and AS).

DS - Data Strobe. For a write cycle, indicates valid data from the processor is on the AD bus. In a read cycle, indicates when data should be placed on the AD bus. Write data should be latched during (phase 2 and DS), read data should be placed on the bus during (phase 1 and DS).

R/W - When high indicates a memory read operation. When low indicates a memory write cycle.

C/D - When high the processor is accessing code memory. When low the cycle is to data memory.

RDY - A high indicates to the processor that it may finish the current memory cycle. While low the processor will wait with the control lines active. Should be synchronized to the leading edge of phase 2. Sampled at the end of phase 2 to determine whether to wait during the next bus cycle.

FIGURE 2.3A

3. PHYSICAL IMPLEMENTATION

The primary tools used in the assembly of the chip were LAP and the SIMULA language that LAP was built upon. Extensive use was made of the standard SSP software and THINGS which is above LAP. Specifically, POINTS and VECTORS were used to hold positions of connection points and references. In addition several routines were written to aid in the assembly.

From the start of the physical layout much care was taken in the referencing of cells or blocks to other cells so that if one had to be moved the rest would follow. This simple trick proved invaluable in adjusting the design.

3.1 SPECIALIZED ROUTINES

The specialized routines were written as a one shot piece of code for GR chip assembly. After examining the routines it appears that they could be used for other designs.

3.1.1 RIVERFRED

One of the major problems in the assembly was connecting a set or vector of points to another set that were askew. Two river routers were written - one for points in the x-axis(riverfredx) and another for points in the y-axis(riverfredy). Given more time a general river router could have been written.

Riverfred inputs are 1) the amount of separation desired between adjacent wire, 2) the distance from the bottom set of points where a jog is allowed to start and 3) the two vectors of points that are to routed between. Riverfred assumes that the routing layer has selected.

Riverfred will jog the routed paths as necessary to connect the points. In the layout in the appendix one can see what riverfred did on the wires from the data path to the output tristate pads.

The error checking that is done is only to insure that the two vectors of points to be router are of equal length. Addition error handling could have been done if more time was available.

3.1.2 PLAAND

This is an AND plane SIMULA class of a pla that was adapted from Dave Johannsens pla generator. This routine is of specific interest in that it was designed to grow or stretch to meet its connective cell. This was used for the decoder section for the data path. The need for stretching was to mate with the input control line buffers for the data path.

The inputs to PLAAND are 1) the number of inputs, 2) the title of the file which contains the personalization for the AND plane, 3) a vector of output points which determines the stretchings and number of outputs, and 4) an integer to determine how much of the plaand is to be generated.

The plaand is a SIMULA object that has several parameters that can be examined. They are: 1) its height, 2) its width, 3) its input points and 4) its output points. These can be used to adjust the surround to it. An example is to use riverfred to connect its outputs to the input of the data path.

The PLAAND first determines its size and number of outputs by examining the output vector of points. This is done by stepping though the output points, checking adjacent points for minimum separation and creates a new point for each output in relation to the vector of output points.

3.2 DATA PATH

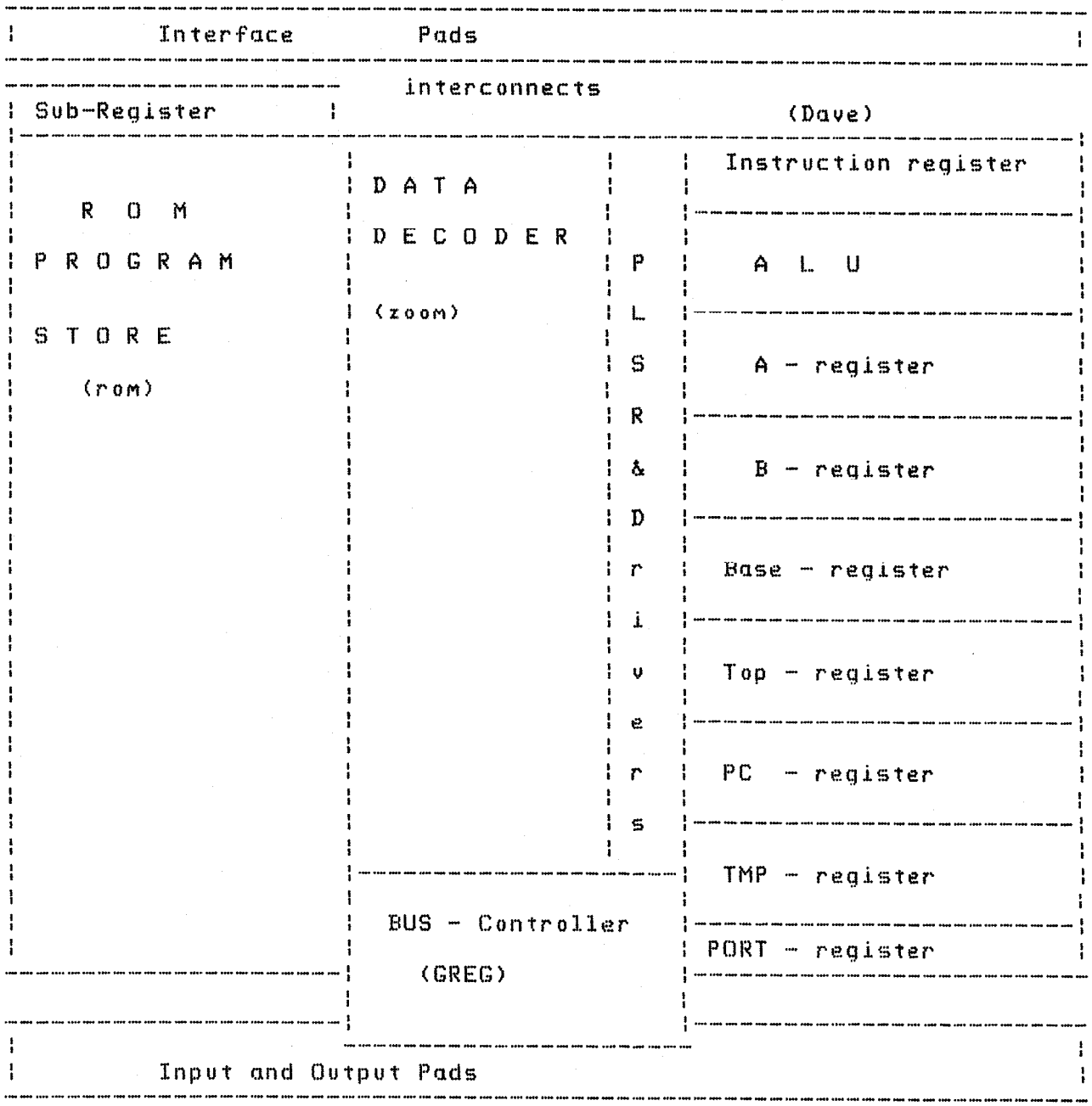
The DATA PATH was created using Bristle Blocks by the renowned chip designer Dave Johannsen for the GR machine. Dave provided the CIF file, a file of connection points and logic equations for the data path. Please see the appendix for the input to Bristle Blocks.

The data lines for the data path run from top to bottom while the control signals are from left to right, except for decode lines for the busses. There are two data buses - one (the A bus) for reading and writing from the registers to memory or thru the Alu, and two (the B bus) for reading to the ALU.

3.3 FLOOR PLAN

The floor plan for the GR machine is represented below. In addition there is a mixed notation diagram in the appendix. The approximate size is 120 by 170 mils.

In the LAP code for the GR machine each major block was labeled with an interesting identifier. On the floor plan each block has its unique and individual name in parenthesis. To correlate the floor plan with the LAP code use the interesting identifiers.



3.3 PRIMITIVE CELLS and MAJOR BLOCKS

Primitive Cells

Subprimcell -> This is a single bit refreshed subroutine latch cell for the subroutine register. It is repeated 6 times for the register. The input data lines for the rom are part of this cell. in addition this cell contains a reset line for the input lines.

Subendcell -> This is a single bit subroutine cell that is always set on. It is used for the low order bit of the subroutine latch.

Plaandinput -> This cell defines the primitive input for the decoder.

Plaandcell -> This cell define the hierarchical cell for the center of the plaand.

Plaandtpullup -> This is the top cell for the pullup for the plaand.

Plaandbpullup -> This is the bottom cell for the pullup for the plaand.

Major Blocks

ROM --> This block is the encoder micro store for the chip. It is connected to subregister block,decoder block , the I/O pads , and the data path. The rom has seven inputs, 74 minterms, and 34 outputs.

SUBREGISTER -> This block contains the seven bit register for subroutine calls in the micro code. It interfaces the control rom and the data path.

TOPBUZ --> This cell contains the routing data from the rom block to the pla ZOOM.

ZOOM --> This cell is the decoder for the data path.

PASCALCHIP -> This cell is the cif file that contains the data path. In the LAP description it is a box that is used as an outline for the data path. It is created optionally depending on the input request.

DAVER --> This cell calls the pascal chip with rotation to

aline to the rest of the chip.

DAVE --> This cell calls daver with translation to position the the data path on the GR chip.

CONGEORGE --> This block defines the interconnection between the decoded ZOOM and the data path dave.

ELBOTTOMPADS --> This contains the description of the bottom pads and there interconnect to DAVE. In addition it contains the major power routing.

ELTOPPADS --> This block contains the definition of the top pads and there corresponding connections to the chip. Also the ground bus routing is partially done here.

LEFTBUZ --> This block defines the interconnections around the rom on the left hand side of the chip.

WOOPEE --> This is a catch all cell that hold the super buffers and miscellaneous connections.

GREGANDRICKYS-CHIP -> This block calls all the other blocks.

3.4 CHIP ASSEMBLY

This section will give a walk thru the LAP code with special attention given to the major block cells and how the major blocks were interconnected. The discussion will begin at the most important point of interest that is the first executable statement. At the beginning of first executable code there is request from the terminal whether to read the CIF file that contains the data path. This is done so that a small CIF file can be generate to test with. In addition requests are made for the personalization of the rom, decoder and bus interface.

The first thing that is done is to draw the rom. The origin of the chip is center at the origin of the roms zero-zero. Next points (that is points defined in THINGS) are set up to refer to the edge of the rom for further use.

The next major block that is define is the subregister cell. It components are placed a set distance from the from inputs since the subcells it uses contain part of the paths for its input. Also a set of vectors for the input of the rom and the subregister were created and passed to riverfred for connection. Next a vector of points for the top of the subregister were created called SUBTOPPOINTS for interconnection to later parts. Also a real variable TOPY was set to the max height for later use.

Next which is not a cell create but a suite of vectors are created that contain point from the data path - in the lap code it is called Dave. These points are translated from dave's 1/2 lambda units to 1 lambda units and also rotated. In the LAP there is a heading called DAVE'S POINTS which precede the points.

Next the topbuz is created which contains major i/o lines from the the rom to connection points to the other blocks. The points are saved as vectors. In addition the power and ground lines for the rom were widen in this cell. In addition a global variable called ZOOMX was set up to the edge of all lines bordering on the x-axis. Finally in this cell lines were routed based on ZOOMX from the outputs of the rom to the inputs plaand ZOOM.

Next the ZOOM is created with its edge on ZOOMX, ZOOM is the decoder part. It is important to note that by changing ZOOMX it adjusts ZOOM and the wires too it.In addition ZOOMX is adjusted to the edge of ZOOM and points are set up for top corner of ZOOM.

The data path is now created with two cells DAVER and DAVE. The reason for this is to compensate for coordinate translations between the data path CIF and the GR engine.

CONGEORGE is now created he defines the interconnects between ZOOM and DAVE(data path). Connection that need to go to other modules are marked in vectors of points.

GREG is created next in relation to ZOOM. GREG is the bus controller module. He also creates wires from himself to the lines created in topbuz. Note that the wire are connected to predefined set of points.

The bottom pads are now created with the interconnection to DAVE and referenced to DAVE. Therefore if DAVE moves, the bottoms pads move. The name of the bottom pad block is - you guest it - ELBOTTOMPADS. Riverfred was used to connect the connection to the bottom pads to the i/o lines of dave. Also the power routing is down in this cell.

The top pads are now define defined in ELTOPPADS which are referenced to TOPY which was used in TOPBUZ. The interconnection from this block to the rest of the chip is defined in this cell.

Next the connection on the left hand side of the chip are defined in LEFTBUZ. This block is referenced to the rom. Following this cell is the catch all for the miscellaneous stuff called WOOPEE. It is generally in reference to the other cells for easy adjustment.

The last cell calls all the others which is called GREGANDRICKYS-CHIP. There were some cells left out of this discussion which were title cells and generally not important.

4. DESIGN AIDS

4.1 MICRO ASSEMBLER

The Micro Assembler is a SIMULA language program that will parse a micro symbolic and assemble a textual representation of the micro code. This textual representation can then be used by the program that assembles the LAP description of the GR chip controlrom. In addition the micro assembler will supply a class set for the GR simulator to be used in testing the micro code and the GR chip in a functional simulation.

4.1.1 Micro Assembler Instruction Set

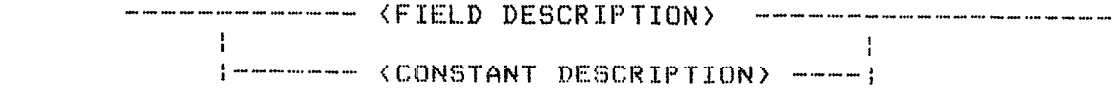
The Micro Assembler instruction set is defined below in a syntax graph form. Please note that the text ⟨EOL⟩ means end of line, and that all the tokens before the ⟨EOL⟩ must be on one line. Comments are allowed on the line provided that they are the last entry, and are preceded by the character '!'. Generally each statement is contained on one line and it is assumed by the micro assembler that there is one and only one statement per line.

Assembler Instructions

Declarations - This defines the declarations for the micro processor.

Micro Code Section - This section defines the instruction for the micro processor.

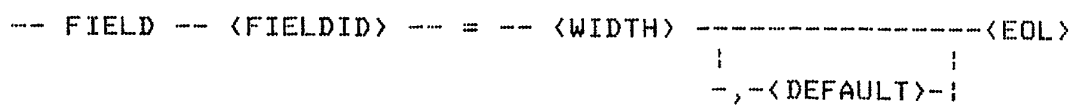
Declarations



Field Description - This defines control fields.

Constant Description - This defines a fixed constant for a specific field that can be used in the micro code instruction.

Field Description

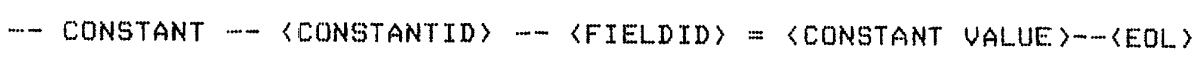


Fieldid - This is the unique identifier of this field. It will be used later in the constant definitions.

Width - This is the number of bit in this specific field. Please note that currently there is no on the constant parameter against this width.

Default - This defines the default integer value for this field.

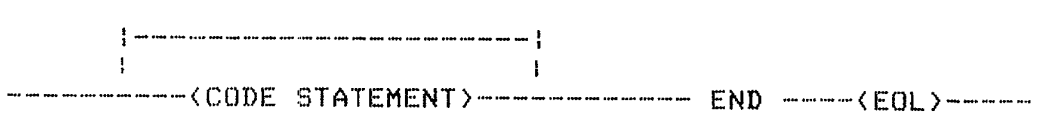
Constant Description



Constantid - This is the unique identifier for this constant. Note that the fieldid is the field that this constant is defined for.

Constantvalue - This is the integer value for this constant.

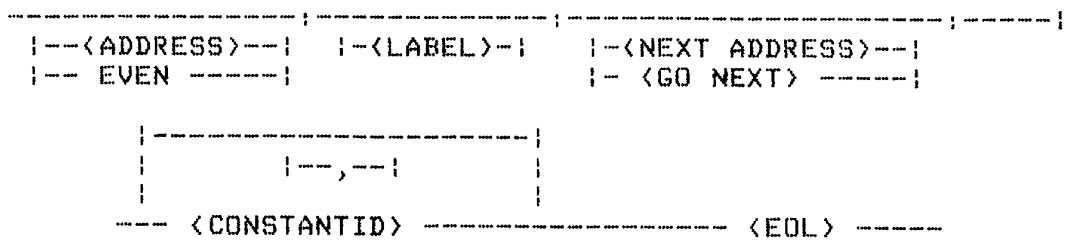
Micro Code Section



Reserve Part - This is an integer that defines the code address from 0 to it for special uses. That is, all automatic address assignments will be above this address.

Code Statement - This defines the actual code statement which is limited to one per line. The loop above the statement allows as many code statements as desired provided that each instructions is on one and only one line.

Code Statement



Address - This integer defines the address for this instruction. If it is left out the assembler will assign an address above the reserve code area.

Even - This tells the assembler to put this address on an even code boundry. Note that you may not assign an address when this is used.

Label - This is a unique label identifier for this instruction. Note that this is an optional quantity.

Next Address - This integer defines the next address. If it is left out the next available code address is used.

Go Next - This is a LABEL identifier that is to be used for the next address.

4.1.2 SIMULATOR

Initially we had planned to do functional testing via a simulator that was partially written at the same time as the micro assembler . Generally the simulator would take the personalization file for the rom and a coded module of the data path, subregister latch, and miscellaneous lines to form a functional simulator. Extensive user interaction was planned to test the chip via the simulator.

During the time of writing the simulator the layout of the chip was also going on. It was apparent after a short time that we did not have enough time to finish the simulator. However, the idea of having the microcode assembler generate data for the simulator will be used for the GR2 machine.

5. HOW WE WOULD DO IT AGAIN

We are currently planning to implement the GR2, a similar machine in architecture though probably not even close to the layout of the GR. We had several problems from the start in that we used two DA systems that did not mesh cleanly, fixed pitch cells and an underestimation of the magnitude of the wiring management problem. For the GR2 we are planning to use the NEW SSP tools and strategy. Part of the ideas follow.

First each cell would be design as a procedure or routine some what like the ones in bristle blocks. They would be able to grow or stretch to match surrounding environments and able to adapt to different power and loading(fan out) requirements based on there surroundings. In addition timing information could be ask for from the cells input to output for the worst case. This is a lot of stuff for a cell to be cognizant of. I am currently looking at creating a sticks editing system to generate the cells and a compaction algorithm to generate the parts for stretching.

Now to connect the cell we would define connection functions that would create instances of the cells and communicate among then for power, and loading. In addition if the cells could not match directly a connection cell would be created to hook them together.

On the next design we plan to spend considerable time in top down design with wiring management considerations and bottom up design for the cell. Iteration over the top down and bottom up will aid in the design in addition to planning for the wiring.

A P P E N D I X ------- A

GRAPHIC PICTURE OF THE GR CHIP

CALTECH 1979
STACK DATA ENGINE
R C MOSTELLER
G EFLAND

A P E N D I X ------- B

LAP DESCRIPTION

```
BEGIN
    EXTERNAL CLASS things,lap281,def281;
    EXTERNAL INTEGER PROCEDURE imin,imax,hash,scanint;
    EXTERNAL REAL PROCEDURE rmin,rmax;
    EXTERNAL BOOLEAN PROCEDURE jsys;
    EXTERNAL LONG REAL PROCEDURE scanreal;
    EXTERNAL TEXT PROCEDURE rest,upcase,frontstrip,conc,skip,scanto;
    EXTERNAL TEXT PROCEDURE today,daytime,lowcase,initem;
    EXTERNAL PROCEDURE enterdebug;

    def281 BEGIN
        INTEGER i,j; REAL x,y;
        INTEGER zoomx;
        INTEGER bluebuzsep;
        INTEGER topy;
        BOOLEAN romguts;
        BOOLEAN zoomguts;
        BOOLEAN gregguts;
        REF(pla) greg;
        REF(pla) rom;
        REF(plaand) zoom;
        ref(vector) zorzk;
        REF(vector) subtoppnt,subbottompnt;
        REF(vector) buscntlpnt;
        REF(vector) irportpd;
        REF(vector) portpd;
        REF(vector) ccpnt;
        REF(vector) allportid;
        REF(vector) threetops;
        REF(point)  nasource;
        REF(point)  topgndpnt;
        REF(point) gregpnt;
        REF(point) edgerompnt;
        REF(point) edgezoompnt;
        REF(point) edgedavepnt;
        REF(point) enableport;
        REF(point) loadport;
        REF(point) portenable;
        REF(point) portload;
        REF(point) plsrout1;
        REF(point) plsrin1;
        REF(point) ph1davepnt;
        REF(point) msbpnt;
        REF(point) zeropnt;
        REF(point) davesize;
        REF(point) ph2romoutpnt;
        REF(point) rdypnt;
        REF(point) rdytpnt;
        REF(point) romgndpnt;
        REF(point) subvddpnt;
        REF(point) subgndtpnt;
        REF(point) subgndbpnt;
        REF(point) subph2pnt;
        REF(point) subenapnt;
        REF(point) subresetpnt;
        REF(vector) lsb;
        REF(point)topline;

        REF(vector) v; ! scratch;
        REF(vector) pascpnt;
```

```
REF(point)  pnt; ! scratch;
REF(point) subreadpnt,subwritepnt;
REF(point)  leftpnt,rightpnt;
BOOLEAN     breadcif;
BOOLEAN     debug;

PROCEDURE laplibstuff;
BEGIN
    INTEGER i;
    ! This file defines the following LAP symbol: PadTriState;
    DEFINE("PADTRISTATE");
    LAYER(METAL);
    WIRE(3,1.5,3.5).X(58.5).Y(24.5);
    BOX(57,26,61,32);
    WIRE(4,10,33).Y(14).X(27);
    WIRE(4,26,33).Y(23).X(48);
    BOX(50,19,54,25);
    BOX(40,29,54,35);
    !  GROUND;
    WIRE(4,2,31).Y(75);
    BOX(4,49,16,57);
    BOX(20,49,32,57);
    BOX(36,49,48,57);
    BOX(52,49,64,57);
    WIRE(4,18,42).Y(64);
    WIRE(4,34,42).Y(64);
    WIRE(4,50,42).Y(64);
    WIRE(8,68,16).Y(90);
    !  VDD;
    WIRE(8,166,4).Y(102);

    WIRE(4,10,73).Y(91).X(27);
    WIRE(4,26,73).Y(83).X(42);
    BOX(44,81,54,87);
    BOX(40,71,54,77);
    !  PAD;
    BOX(90,26,144,80);
    BOX(90,15,94,26);
    BOX(115,15,119,26);
    BOX(140,15,144,26);
    BOX(144,26,154,30);
    BOX(144,51,154,55);
    BOX(144,76,154,80);
    BOX(140,80,144,91);
    BOX(115,80,119,91);
    BOX(90,80,94,91);
    BOX(89,11,95,15);
    BOX(114,11,120,15);
    BOX(139,11,145,15);
    BOX(154,25,158,31);
    BOX(154,50,158,56);
    BOX(154,75,158,81);
    BOX(139,91,145,95);
    BOX(114,91,120,95);
    BOX(89,91,95,95);
    LAYER(GLASS);
    BOX(94,30,140,76);

    LAYER(POLY);
    WIRE(2,1,24).X(14).Y(46);
```

```
WIRE(2,36,14).X(57);
WIRE(2,42,16).Y(21).X(22).Y(79).X(14).Y(60);
WIRE(2,6,28).Y(83).X(30).Y(27).X(38).Y(79).X(32);
WIRE(2,27,85).Y(96).X(32).Y(97);
WIRE(2,1,100).X(26);
WIRE(2,36,100).X(43);
WIRE(2,36,92).X(57);
WIRE(2,42,90).Y(79).X(46).Y(28);
WIRE(2,55,34).X(75).Y(17).X(152).Y(89).X(75).Y(72).X(55);
WIRE(2,55,24).X(69).Y(9).X(160).Y(97).X(69).Y(82).X(55);
BOX(50,19,54,25);
BOX(50,29,54,35);
BOX(50,71,54,77);
BOX(50,81,54,87);

LAYER(IMPLANT);
BOX(34,11,58,17);
BOX(34,89,58,95);
BOX(34,97,44,103);

LAYER(DIFFUSION);
BOX(64,0,170,28);
BOX(64,78,170,106);
BOX(64,28,92,78);
BOX(142,28,170,78);
BOX(57,26,64,32);
BOX(44,11,56,18);
BOX(29,16,33,19);
BOX(33,17,36,19);
WIRE(4,38,17).Y(9).X(62);
WIRE(4,42,27).Y(25);
WIRE(4,46,24).Y(20);
BOX(4,29,44,45);
BOX(4,61,44,77);
BOX(16,45,44,61);
BOX(44,36,52,70);
BOX(44,29,49,36);
WIRE(4,2,31).Y(96);
WIRE(6,4,100).X(22);
WIRE(2,24,103).X(30);
WIRE(2,36,103).X(41);
BOX(36,98,42,102);
BOX(29,86,33,89);
BOX(33,86,36,88);
WIRE(4,38,88).Y(96).X(62);
BOX(44,88,56,94);
BOX(44,81,48,88);

LAYER(CUTS);
BOX(1,30,3,34);
BOX(1,40,3,44);
BOX(1,51,3,55);
BOX(1,62,3,66);
BOX(1,72,3,76);
BOX(9,30,11,34);
BOX(9,72,11,76);
BOX(17,41,19,45);
BOX(17,51,19,55);
BOX(17,61,19,65);
BOX(25,30,27,34);
```

```
        BOX(25,72,27,76);
        BOX(33,41,35,45);
        BOX(33,51,35,55);
        BOX(33,61,35,65);
        BOX(41,30,43,34);
        BOX(41,72,43,76);
        BOX(49,41,51,45);
        BOX(49,51,51,55);
        BOX(49,61,51,65);
        BOX(51,30,53,34);
        BOX(51,20,53,24);
        BOX(58,27,60,31);
        BOX(51,72,53,76);
        BOX(51,82,53,86);
        BOX(45,82,47,86);
        BOX(65,39,67,43);
        BOX(65,47,67,51);
        BOX(65,55,67,59);
        BOX(65,63,67,67);
        BOX(69,39,71,43);
        BOX(69,47,71,51);
        BOX(69,55,71,59);
        BOX(69,63,71,67);
        BOX(90,12,94,14);
        BOX(115,12,119,14);
        BOX(140,12,144,14);
        BOX(155,26,157,30);
        BOX(155,51,157,55);
        BOX(155,76,157,80);
        BOX(90,92,94,94);
        BOX(115,92,119,94);
        BOX(140,92,144,94);
        BOX(165,6,167,10);
        BOX(165,16,167,20);
        BOX(165,26,167,30);
        BOX(165,36,167,40);
        BOX(165,46,167,50);
        BOX(165,56,167,60);
        BOX(165,66,167,70);
        BOX(165,76,167,80);
        BOX(165,86,167,90);
        BOX(165,96,167,100);

        BE(30,14);
        BE(30,91);
        BS(33,103);
        ENDDEF;


! This LAP file contains definitions of symbols:
      PadOut,  PadBlank,  PadOut,  PadClockedOut
        DEFINE("Strap");
        Layer(Metal);
        box(0,0,4,6);
        box(4,1,14,5);

        Layer(Cuts);
        box(1,1,3,5);
        EndDef;
```

```
DEFINE("ayatollah");
Layer(metal);
box(26,-80,80,-26); !the pad;
Layer(glass);
box(30,-76,76,-30); !the overglass cut;
EndDef;

DEFINE("PadBlank");

Layer(metal);
wire(8,4,-4).x(102);          !Vdd line;
wire(8,16,-102).x(90);        !ground line;
box(26,-80,80,-26); !the pad;

Layer(glass);
box(30,-76,76,-30); !the overglass cut;

EndDef;

DEFINE("Basic Pad");
draw("PadBlank",0,0);                  !make the pad, vdd, gnd and glass
Layer(Diffusion);
wire(28,14,-14).x(92).y(-92).x(14).y(-42); !make the huge diffusion;
wire(4,9,-106).y(-131).x(15); !a Vdd source for pullups.;
wire(2,18,-130).y(-144).x(30); !an inverter;
box(29,-145,45,-137);         !connections;
box(11,-121,18,-109);
box(10,-119,25,-115);
box(16,-140,19,-136);
bn(14,-139);                  !connection to the output, becoming a SB;

Layer(Poly);
wire(2,14,-134).y(-108); !gates on pullups.;
wire(2,16,-123).x(27).y(-119).x(78);
!and a gate on the next to last;

Layer(Metal);
box(19,-119,25,-110);         !a last stage connection for a pullup.;
wire(4,23,-121).y(-131).x(33); !and its corresponding gate.;

Layer(Cuts);
box(20,-113,24,-111);
box(20,-118,24,-116);

box(30,-132,34,-130);

Layer(Poly);
box(19,-114,25,-110);
wire(2,24,-109).y(-101).x(9).y(-10).x(97).y(-101).x(82).y(-109);
!the final stage gate, pull up.;
box(81,-114,87,-110);

Layer(Cuts);
box(82,-113,86,-111);

Layer(Metal);
box(81,-114,87,-110);
wire(4,83,-116).y(-131).x(73);

Layer(Cuts);
```

```
box(72,-132,76,-130);
FOR i:=-1 STEP 1 UNTIL 1 DO
box(i*10+51,-124,i*10+55,-122);

Layer(Diffusion);
wire(2,78,-116).x(87);
box(29,-131,77,-115);
box(71,-133,77,-131);
box(29,-133,35,-131);
box(36,-115,42,-110);
box(64,-115,70,-110);

Layer(Cuts);
box(30,-113,34,-111);
box(37,-113,41,-111);
box(65,-113,69,-111);
box(72,-113,76,-111);

Layer(Metal);
box(29,-114,42,-110);
box(64,-114,77,-110);

Layer(Poly);
box(29,-114,35,-110);
wire(2,34,-109).y(-95).x(18).y(-18).x(88).y(-95).x(72).y(-109);
!the final stage pull down.;
box(71,-114,77,-110);

Layer(Cuts);
FOR i:=0 STEP 1 UNTIL 9 DO
box(10*i+6,-5,10*i+10,-3);

Layer(Metal);
box(49,-121,57,-106);        !the ground connections for a few stages;
box(40,-125,66,-121);

!the straps connecting the pad to the last level transistors.;
DRAWMX("STRAP",94,-81);
DRAWMX("STRAP",94,-56);
DRAWMX("STRAP",94,-31);
DRAWROT("STRAP",75,-12,0,-1);
DRAWROT("STRAP",50,-12,0,-1);
DRAWROT("STRAP",25,-12,0,-1);
draw("STRAP",12,-31);
draw("STRAP",12,-56);
draw("STRAP",12,-81);

Layer(Cuts);
!connect one side of the huge diffusion to ground.;
FOR i:=0 STEP 1 UNTIL 3 DO
BEGIN
    box(i*8+39,-101,i*8+43,-99);
    box(i*8+39,-105,i*8+43,-103);
END;

Layer(Poly);
wire(2,46,-141).x(27).y(-127).x(78);

Layer(Implant);
!make the pullups;
```

```
box(11.5,-122.5,16.5,-107.5);
box(11.5,-134.5,16.5,-127.5);
EndDef;      !End of Basic Pad.;

DEFINE("PadOut");
draw("Basic Pad",0,0);

Layer(Diffusion);
wire(4,97,-108).y(-131).x(89);
wire(2,88,-134).y(-144).x(78);
box(88,-121,95,-109);
box(89,-140,90,-136);
box(61,-145,77,-137);
box(40,-139,66,-135);
BN(92,-139);

Layer(Poly);
wire(2,53,-144).y(-141).x(78);
wire(2,92,-133).y(-108);
wire(2,80,-127).x(90);

Layer(Metal);
wire(8,53,-129).y(-131);
box(40,-139,66,-135);

Layer(Cuts);
box(41,-138,45,-136);
box(51,-138,55,-136);
box(61,-138,65,-136);

Layer(Implant);
Box(89.5,-122.5,94.5,-107.5);
box(89.5,-134.5,94.5,-127.5);
EndDef;


define("PadIn");
draw("PadBlank",0,0);        ! Pad, VDD, GND and glass cut;
layer(metal);
box(26,-26,30,-12); ! Wire from pad;

layer(cuts);
box(27,-17,29,-13);

layer(diffusion);
box(26,-18,30,-12);
wire(2,27,-11).x(96).y(-105);

box(89,-99,95,-86); ! Diffusion for lightening arrestor;
box(12,-102,89,-84);

layer(poly);                 ! Lightening arrestor gate;
wire(2,93,-85).y(-89).x(20).y(-96).x(93).y(-99);

rb(92,-102);                 ! Connect gate to ground;

layer(cuts);                 ! Connect difusion to ground;
FOR i:=13 STEP 10 UNTIL 83 DO box(i,-101,i+4,-99);

enddef;
```

```
! This file defines the LAP symbols:
        NonInvertingSingleSB,  NonInvertingSBPair,  InvertingSBPair;
            DEFINE("LCUT");                         !LONG CUT (2X4);
            LAYER(GREEN);
            BOX(0,0,4,6);
            LAYER(CUTS);
            BOX(1,1,3,5);
            ENDDEF;


            DEFINE("BUFBLK");           !BASIC SUPER BUFFER BUILDING BLOCK;

            LAYER(BLUE);
            WIRE(4,0,16).X(16);         !VDD;
            wire(6,1,43).x(15);

            LAYER(GREEN);
            WIRE(2,0,-9).Y(28);         !CENTER (GND) EXTENSION;
            WIRE(2,8,-9).Y(7);          !"OR" EXTENSION;
            WIRE(2,10,9).Y(21);         !PULLUP CHANNELS;
            BOX(1,6,6,10);                      !INPUT CHANNEL...;
            BOX(6,6,7,8);
            BOX(1,17,6,25);                     !BUFFERED CHANNEL...;
            BOX(6,22,8,25);
            LAYER(IMPLANT);
            BOX(7,7,13,15);                     !PULLUP IMPLANTS;
            BOX(7,17,13,23);
            LAYER(RED);
            WIRE(2,3,1).Y(11);          !INPUT GATE;
            WIRE(2,13,9).Y(12).X(7).Y(15).X(3).Y(27).X(4).Y(44);
            WIRE(2,8,20).X(13).Y(24).X(12).Y(44);
            BOX(7,9,12,11);

            BE(9,6);                            !????;
            BE(9,24);
            GB(11,16);
            draw("LCUT",-2,29);
            draw("LCUT",-2,40);
            draw("LCUT",14,40);

            ENDDEF;


            DEFINE("NonInvertingSingleSB");             !NON INVERTING SINGLE INF
            draw("BUFBLK",0,0);

            LAYER(RED);

            WIRE(2,3,-1).Y(-9);

            LAYER(GREEN);

            BOX(2,29,7,35);
            WIRE(2,-1,36).X(8).Y(45);
            WIRE(2,10,42).X(13);

            LAYER(IMPLANT);
            BOX(9,39,15,45);
```

```
    ENDDEF;

END of laplibstuff;

CLASS Plaand(input,ptvec,codefile,PLAguts); VALUE codefile;
!-------------------------------------------------------------------;
INTEGER input,PLAguts;REF(vector)ptvec; TEXT codefile;
! IF PLAguts ) 7 THEN draws everything   ;
BEGIN
    INTEGER i,j;
    INTEGER pterm;
    CHARACTER c;
    REAL x,y,maxy;
    REF(vector) outzip;
    TEXT inputcolumnname,groundcolumnname,planame;
    REF(infile) filin;
    REF(point)p;

    REF(point) PROCEDURE output(i);VALUE i;INTEGER i;
    BEGIN
        IF i ) pterm OR i( 1 THEN x:=x/0;
        output:-outzip.val(i) QUA point;
    END;

    REAL PROCEDURE plawidth;
    BEGIN
        REAL x;

        x:=14*input+15.5+16;
        x:=x+21.5;
        plawidth:=x;
    END;

    REAL PROCEDURE plaheight;
    BEGIN
        REAL y;
        INTEGER i;
        y:=maxy;
        plaheight:=y;
    END;

    REF(point) PROCEDURE plainput(n); INTEGER n;
    plainput :- NEW point(14*n,0);

    REF(rectangle) PROCEDURE mbb;
    BEGIN
        REAL y;

        y:=-.5;
        mbb:-NEW rectangle(NEW point(-11,y),
        NEW point(
        14*input+15.5
        +10.5,plaheight+y));
    END;

    PROCEDURE buildopnt;
    BEGIN
        BOOLEAN b;
        ! lets build output points from ptvec;
        pterm:=ptvec.length;
```

```
    outzip:-NEW vector;
    x:=14;
    FOR i:=1 STEP 1 UNTIL input DO
    x:=x+14;
    x:=x+16;
    ! now to generate points;
    b:=ptvec.val(1) QUA point.y < ptvec.val(2) QUA point.y;
    j:=IF b THEN 1 ELSE pterm;
    y:=60.5-8-3.5;
    FOR i:=1 STEP 1 UNTIL pterm DO
    BEGIN
        p:-ptvec.val(j) QUA point;
        IF p.y > y THEN y:=p.y;
        outzip.append(NEW point(x-.5+8.5,y));
        y:=y+7;
        j:=IF b THEN j+1 ELSE j-1;
        IF i NE pterm AND (i//8)*8=i AND i<pterm-4 THEN
        y:=y+7;
    END;
    ptvec:-NONE;
END of buildopnt;

define("PLAandinput");
layer(green);
wire(2,-2,16).y(36);
wire(2,7,14).y(44);
wire(2,1,50).y(54.5);
box(-2,39,8,35);
box(-2,17,8,13);
layer(red);
wire(2,4,8).y(18);
wire(2,0,22).x(4).y(46.5).xy(5,47.5).y(53).xy(4,54).y(55);
wire(2,-2,38).y(46.5).xy(-3,47.5).y(53).xy(-2,54).y(55);
box(-5,39,-1,29);box(-2,35,1,29);
box(-5,23,-1,13);box(-2,23,1,17);
layer(implant);
box(-4.5,39,.5,13);
layer(metal);
wire(4,-8,26).x(8);
box(-8,52,10,42);
gb(-1,26);gb(1,50);gb(8,44);
be(0,8);bw(1,15);bw(1,37);
enddef;

define("PLAandcell");
layer(green);
wire(2,0,-7.5).y(0.5);
gb(7,-3.5);
layer(poly);
wire(2,3,-6.5).y(1.5);
wire(2,-3,-6.5).y(1.5);
layer(metal);
wire(3,-7,-3.5).x(7);
enddef;

define("PLaandcont");
be(-6.5,-3.5);
layer(poly);
wire(2,-2,-3.5).x(8.5+16);
layer(metal);
```

```
wire(4+16,8+3.5,-7.5).y(0.5);
enddef;

define("PLAandtpullup");
layer(green);
wire(2,-.5,3.5).x(-9.5).y(-3.5);
layer(red);
box(-7.5,6.5,-1.5,.5);
layer(metal);
wire(3,-.5,3.5).x(8.5);
layer(implant);
box(-9,6,8,-4);
bw(.5,3.5);
gb(-9.5,-2.5);gb(7.5,3.5);
enddef;

define("PLAandbpullup");
layer(green);
wire(2,-9.5,-3.5).x(7.5);
layer(red);
box(.5,-.5,6.5,-6.5);
layer(metal);
layer(implant);
box(-9,4,8,-6);
bw(8.5,-3.5);
gb(-9.5,-2.5);
enddef;

! define the basic unencoded inputcolumn;
buildopnt;
inputcolumnname:=conc(codefile,"inputcolumn");
define(inputcolumnname);
draw("PLAandinput",0,-8);
y:=60.5-8;
FOR i:=1 STEP 1 UNTIL pterm DO
BEGIN
    p:=outzip.val(i) QUA point;
    IF p.y+3.5 > y THEN
    BEGIN
        layer(green);
        wire(2,i,.5+y-7).y(p.y+3.5-7.5);
        layer(poly);
        wire(2,3+i,1.5+y-7).y(p.y+3.5-6.5);
        wire(2,-3+i,1.5+y-7).y(p.y+3.5-6.5);
    END;
    y:=p.y+3.5;
    draw("plaandcell",i,y);
    y:=y+7;
    IF (i//8)*8=i AND i<pterm-4 THEN
    BEGIN
        draw("PLAground",i,y-3.5);
        y:=y+7;
    END;
END of for;
draw("PLAground",i,y-3.5);
enddef;

!generate the body of the pla;
planame:=conc(codefile,"pla");
define(planame);
```

```
y:=60.5-8;
FOR i:=1 STEP 1 UNTIL pterm DO
BEGIN
     p:-outzip.val(i) QUA point;
     y:=p.y+3.5;
     IF (i//2)*2<i THEN
     draw("PLAandtpullup",.5,y-7) ELSE
     draw("PLAandBpullup",.5,y);
     y:=y+7;
     IF (i//8)*8=i AND i<pterm-4 THEN y:=y+7;
END;
x:=14;
FOR i:=1 STEP 1 UNTIL input DO
BEGIN
     IF plaguts > 3 THEN
     draw(inputcolumnname,x,0);
     x:=x+14;
END;
y:=60.5-8;
FOR i:=1 STEP 1 UNTIL pterm DO
BEGIN
     p:-outzip.val(i) QUA point;
     IF p.y+3.5 > y THEN
     BEGIN
          layer(metal);
          wire(4+16,+8+x-.5+3.5,0.5+y-7).y(p.y+3.5-7.5);
     END;
     y:=p.y+3.5;
     draw("PLAandcont",x-.5,y);
     y:=y+7;
     IF (i//8)*8=i AND i<pterm-4 THEN
     BEGIN
          draw("PLAtee",x-.5,y-3.5);
          y:=y+7;
     END;
END of for;
draw("PLAtee",x-.5,y-3.5);
x:=x+13.5;
maxy:=y+10;

layer(metal);
wire(4,-9,y+10).y(26-8).x(14*input+2);
box(14*input+9,52-8,14*input+19,42-8);
!wire(4,-9,0-8).y(26-8);

y:=60.5-8;

IF PLAguts > 3 THEN
BEGIN
     layer(green);
     filin:-NEW infile(conc(codefile,".COD"));
     filin.open(blanks(132));
     y:=60.5-8;
     FOR i:=1 STEP 1 UNTIL pterm DO
     BEGIN
          x:=14;
          filin.inimage;
          p:-outzip.val(i) QUA point;
          y:=p.y+3.5;
          FOR j:=1 STEP 1 UNTIL input DO
```

```
            BEGIN
                c:=filin.inchar;
                IF c='1' THEN box(x+1,y-1.5,x+7,y-5.5)
                ELSE IF c='0' THEN box(x-5,y-1.5,x+1,y-5.5);
                x:=x+14;
            END;
            IF ((i+1)//8)*8=(i+1) AND i<pterm-5 THEN y:=y+7;
        END;
        filin.close;
    END;
    enddef;
END of plaand;


PROCEDURE riverfrank(vb,vt);REF(vector) vb,vt;
!-------------------------------------------;
BEGIN
    REF(point) p1,p2,pli,pri;
    INTEGER i,j,k;
    i:=j:=vb.length;
    IF vb.length NE vt.length THEN i:=i/0;
    p1:-vb.val(i)QUA point;p2:-vt.val(i) QUA point;
    WHILE i>=1  AND p1.x > p2.x DO
    BEGIN
        i:=i-1;
        p1:-vb.val(i)QUA point;p2:-vt.val(i) QUA point;
    END;
    i:=i+1;
    WHILE i<=j DO
    BEGIN
        p1:-vb.val(i)QUA point;p2:-vt.val(i) QUA point;
        IF pli == NONE THEN
        pli :- NEW point(p2.x,p2.y-3) ELSE
        IF pli.x-5 >= p1.x THEN
        BEGIN
            pli.y:=p2.y-3;pli.x:=p2.x;
        END ELSE
        BEGIN
            pli.y:=pli.y-5;
            pli.x:=p2.x;
        END;
        wire(2,p1.x,p1.y).y(pli.y).x(p2.x).y(p2.y);
        i:=i+1;
    END;
    i:=1;p1:-vb.val(i)QUA point;p2:-vt.val(i) QUA point;
    WHILE i<=j  AND p1.x < p2.x DO
    BEGIN
        i:=i+1;
        p1:-vb.val(i)QUA point;p2:-vt.val(i) QUA point;
    END;
    i:=i-1;
    WHILE i>=1 DO
    BEGIN
        p1:-vb.val(i)QUA point;p2:-vt.val(i) QUA point;
        IF pri == NONE THEN
        pri :- NEW point(p2.x,p2.y-3) ELSE
        IF pri.x+5 <= p1.x THEN
        BEGIN
            pri.y:=p2.y-3;pri.x:=p2.x;
```

```
        END ELSE
        BEGIN
            pri.y:=pri.y-5;
            pri.x:=p2.x;
        END;
        wire(2,p1.x,p1.y).y(pri.y).x(p2.x).y(p2.y);
        i:=i-1;
    END;
END of riverfank;

PROCEDURE riverfredx(sep,offbottom,vb,vt);VALUE sep,offbottom;
!-------------------------------------------------------------------------------;
REF(vector)vb,vt;INTEGER sep,offbottom;
BEGIN
    REF(point) p1,p2,ps,p;
    INTEGER i,j,k,m,n;
    BOOLEAN debug;
    REF(vector) v,w1,w;
    debug:=FALSE;
    i:=1;j:=vb.length;
    IF vb.length NE vt.length THEN i:=i/0;
    outtext("RIVERFREDX !!!! length=");
    outint(j,4);outtext(",sep=");outint(sep,4); outimage;
    p1:-vb.val(1)QUA point;p2:-vt.val(1) QUA point;
    IF abs(p1.y-p2.y) < sep*2 THEN i:=i/0;
    i:=1;j:=vb.length;
    p1:-vb.val(1)QUA point;p2:-vb.val(j) QUA point;
    IF p1.x > p2.x THEN
    BEGIN
        outtext("   warning vector B inverted!!!");outimage;
        v:-NEW vector;
        k:=j;
        FOR i:=1 STEP 1 UNTIL j DO
        BEGIN
            v.append(vb.val(k));
            k:=k-1;
        END;
        vb:-v;v:-NONE;
    END;
    i:=1;j:=vb.length;
    p1:-vt.val(1)QUA point;p2:-vt.val(j) QUA point;
    IF p1.x > p2.x THEN
    BEGIN
        outtext("   warning vector T inverted!!!");outimage;
        v:-NEW vector;
        k:=j;
        FOR i:=1 STEP 1 UNTIL j DO
        BEGIN
            v.append(vt.val(k));
            k:=k-1;
        END;
        vt:-v;
        v:-NONE;
    END;
    p1:-vb.val(1)QUA point;p2:-vt.val(1) QUA point;
    IF p1.y > p2.y  THEN
    BEGIN
        outtext("   warning dir reversed!!!");outimage;
        v:-vb;vb:-vt;v:-vb;
    END;
```

```
FOR j:=vb.length STEP -1 UNTIL 1 DO
BEGIN
    p1:-vb.val(j)QUA point;p2:-vt.val(j) QUA point;
    IF debug THEN
    BEGIN
        outtext(" j=");outint(j,3);
        outtext(" 1.x=");outint(p1.x,4);
        outtext(" 1.y=");outint(p1.y,4);
        outtext(" 2.x=");outint(p2.x,4);
        outtext(" 2.y=");outint(p2.y,4);
        outimage;
    END;
    IF  p1.x < p2.x THEN
    BEGIN
        w:-NEW vector;w.append(p1);
        ps:-NEW point(p1.x,p1.y+sep+offbottom);
        w.append(ps);
        m:=IF w1 =/= NONE THEN w1.length ELSE -1;n:=1;
        IF debug THEN
        BEGIN
            outtext(" first m=");outint(m,3);
            outtext(" 1.x=");outint(p1.x,4);
            outtext(" 1.y=");outint(p1.y,4);
            outtext(" ps.x=");outint(ps.x,4);
            outtext(" ps.y=");outint(ps.y,4);
            outimage;
        END;
        WHILE n<=m DO
        BEGIN
            p:-w1.val(n) QUA point;
            IF debug THEN
            BEGIN
                outtext("  n=");outint(m,3);
                outtext(" p.x=");outint(p.x,4);
                outtext(" p.y=");outint(p.y,4);
                outtext(" ps.x=");outint(ps.x,4);
                outtext(" ps.y=");outint(ps.y,4);
                outimage;
            END;
            IF ps.y < p.y+sep AND p.x-sep < p2.x THEN
            BEGIN
                ps:-NEW point(p.x-sep,ps.y);w.append(ps);
                ps:-NEW point(p.x-sep,p.y+sep);w.append(ps);
                IF debug THEN
                BEGIN
                    outtext("  new =");outint(m,3);
                    outtext(" ps.x=");outint(ps.x,4);
                    outtext(" ps.y=");outint(ps.y,4);
                    outimage;
                END;
            END;
            n:=n+1;
        END;
        m:=w.length;
        ps:-w.val(1) QUA point;
        FOR n:=2 STEP 1 UNTIL m DO
        BEGIN
            p:-w.val(n) QUA point;
            IF p.x=ps.x THEN
            wire(2,ps.x,ps.y).y(p.y) ELSE
```

```
                wire(2,ps.x,ps.y).x(p.x);ps:-p;
            END;
            w1:-w;
            wire(2,ps.x,ps.y).x(p2.x).y(p2.y);
        END;
    END;
    j:=vb.length;i:=1;w1:- NONE;
    WHILE i<=j DO
    BEGIN
        p1:-vb.val(i)QUA point;p2:-vt.val(i) QUA point;
        IF debug THEN
        BEGIN
            outtext(" i=");outint(i,3);
            outtext(" 1.x=");outint(p1.x,4);
            outtext(" 1.y=");outint(p1.y,4);
            outtext(" 2.x=");outint(p2.x,4);
            outtext(" 2.y=");outint(p2.y,4);
            outimage;
        END;
        IF  p1.x > p2.x THEN
        BEGIN
            w:-NEW vector;w.append(p1);
            ps:-NEW point(p1.x,p1.y+sep+offbottom);
            w.append(ps);
            m:=IF w1 =/= NONE THEN w1.length ELSE -1;n:=1;
            WHILE n<=m DO
            BEGIN
                p:-w1.val(n) QUA point;
                IF ps.y < p.y+sep AND p.x+sep > p2.x THEN
                BEGIN
                    ps:-NEW point(p.x+sep,ps.y);w.append(ps);
                    ps:-NEW point(p.x+sep,p.y+sep);w.append(ps);
                END;
                n:=n+1;
            END;
            m:=w.length;
            ps:-w.val(1) QUA point;
            FOR n:=2 STEP 1 UNTIL m DO
            BEGIN
                p:-w.val(n) QUA point;
                IF p.x=ps.x THEN
                wire(2,ps.x,ps.y).y(p.y) ELSE
                wire(2,ps.x,ps.y).x(p.x);ps:-p;
            END;
            w1:-w;
            wire(2,ps.x,ps.y).x(p2.x).y(p2.y);
            i:=i+1;
        END ELSE
        IF j>= i AND p1.x = p2.x THEN
        BEGIN
            wire(2,p1.x,p1.y).y(p2.y);
            i:=i+1;
        END ELSE i:=i+1;
    END;
END of riverfrezx;

PROCEDURE riverfredy(w,sep,v1,vr);VALUE w,sep;
REF(vector) v1,vr;INTEGER w,sep;
!-------------------------------------------------------------------------------;
BEGIN
```

```
REF(point) p1,p2,pli,pri;
INTEGER i,j,k;
REF(vector) v;
IF vl.length NE vr.length THEN i:=i/0;
i:=1;j:=vl.length;
outtext("RIVERFREDY !!!! length=");
outint(j,4);outtext(",sep=");outint(sep,4); outimage;
i:=1;j:=vl.length;
p1:-vl.val(i)QUA point;p2:-vr.val(i) QUA point;
IF abs(p1.x-p2.x) < sep*2 THEN i:=i/0;
p1:-vl.val(i)QUA point;p2:-vl.val(j) QUA point;
IF p1.y > p2.y THEN
BEGIN
    outtext("   warning vectors inverted!!!");outimage;
    v:-NEW vector;
    k:=j;
    FOR i:=1 STEP 1 UNTIL j DO
    BEGIN
        v.append(vl.val(k));
        k:=k-1;
    END;
    vl:-v;
    v:-NEW vector;
    k:=j;
    FOR i:=1 STEP 1 UNTIL j DO
    BEGIN
        v.append(vr.val(k));
        k:=k-1;
    END;
    vr:-v;
    v:-NONE;
END;
p1:-vl.val(1)QUA point;p2:-vl.val(j) QUA point;
i:=abs(p1.y-p2.y);
p1:-vr.val(1)QUA point;p2:-vr.val(j) QUA point;
k:=abs(p1.y-p2.y);
IF i > k THEN ! sides must be reversed;
BEGIN
    outtext("   warning vectors reversed!!!");outimage;
    v:-vl;vl:-vr;vr:-v;
END;
p1:-vl.val(1)QUA point;p2:-vr.val(1) QUA point;
IF p1.x > p2.x  THEN
BEGIN
    outtext("   warning dir reversed!!!");outimage;
    sep:=-sep;
END;
i:=1;j:=vl.length;
WHILE j>=i DO
BEGIN
    p1:-vl.val(j)QUA point;p2:-vr.val(j) QUA point;
    IF j >= i AND p1.y < p2.y THEN
    BEGIN
        IF pli == NONE THEN
        pli :- NEW point(p1.x+sep,p1.y) ELSE
        IF pli.y-sep >= p2.y THEN
        BEGIN
            pli.x:=p1.x+sep;
            pli.y:=p1.y;
        END
```

```
            ELSE
            BEGIN
                pli.x:=pli.x+sep;
                pli.y:=p1.y;
            END;
            wire(w,p1.x,p1.y).x(pli.x).y(p2.y).x(p2.x);
            j:=j-1;
        END;
        IF j>=i THEN
        BEGIN
            p1:-vl.val(j)QUA point;p2:-vr.val(j) QUA point;
        END;
        IF j>= i AND p1.y = p2.y THEN
        BEGIN
            wire(w,p1.x,p1.y).x(p2.x);
            j:=j-1;
        END;
        p1:-vl.val(i)QUA point;p2:-vr.val(i) QUA point;
        IF j>= i AND p1.y > p2.y THEN
        BEGIN
            IF pri == NONE THEN
            pri :- NEW point(p1.x+sep,p1.y) ELSE
            IF pri.y+sep <= p2.y THEN
            BEGIN
                pri.x:=p1.x+sep;
                pri.y:=p1.y;
            END
            ELSE
            BEGIN
                pri.x:=pri.x+sep;
                pri.y:=p1.y;
            END;
            wire(w,p1.x,p1.y).x(pri.x).y(p2.y).x(p2.x);
            i:=i+1;
        END;
        IF j>=i THEN
        BEGIN
            p1:-vl.val(i)QUA point;p2:-vr.val(i) QUA point;
        END;
        IF j>= i AND p1.y = p2.y THEN
        BEGIN
            wire(w,p1.x,p1.y).x(p2.x);
            i:=i+1;
        END;
    END;
    outtext("end of RIVERFREDY !!!! "); outimage;
END of riverfredy;

PROCEDURE adjustpnty(y);VALUE y;REAL y;
BEGIN
    IF y<leftpnt.y THEN leftpnt.y:=y;
    IF y>rightpnt.y THEN rightpnt.y:=y;
END;
PROCEDURE adjustpntx(x);VALUE x;REAL x;
BEGIN
    IF x<leftpnt.x THEN leftpnt.x:=x;
    IF x>rightpnt.x THEN rightpnt.x:=x;
END;
BEGIN  ! a real cheap way to tell if we have a command line;
    REAL   i,j;
```

```
    TEXT cline;

    outtext("HI THERE = READCIF?");breakoutimage;
    inimage;
    cline:-copy(image);
    breadcif:=sysin.inchar='y';
    IF breadcif THEN
    outtext("readcif") ELSE outtext("no read");outimage;
    outtext("ROM GUTS?");breakoutimage;
    inimage;
    cline:-copy(image);
    romguts:=sysin.inchar='y';
    IF romguts THEN
    outtext("WE HAVE ROMGUTS") ELSE outtext("NO guts");outimage;
    outtext("ZOOM GUTS?");breakoutimage;
    inimage;
    cline:-copy(image);
    zoomguts:=sysin.inchar='y';
    IF ZOOMguts THEN
    outtext("WE HAVE ZOOMGUTS") ELSE outtext("NO guts");outimage;
    outtext("GREG GUTS?");breakoutimage;
    inimage;
    cline:-copy(image);
    GREGguts:=cline.sub(1,1)="y";
    gregguts:=sysin.inchar='y';
    IF GREGguts THEN
    outtext("WE HAVE GREGUTS") ELSE outtext("NO guts");outimage;
    outtext("GUTS UP BUTCH!!!!");outimage;
END;

cifscale:=2;
IF breadcif THEN
readcif("pascal.cif");
laplibstuff;



define("subprimcell");
!-------------------------------------------------------------------------;
BEGIN
    INTEGER x,gy;
    gy:=18;!ground wire;
    ! the origin is at the center or the input to pla;
    layer(blue);
    wire(4,0,0).dx(-21); wire(3,0,0).dx(2);! vdd ;
    wire(4,0,gy).dx(-21); wire(3,0,gy).dx(2);! gnd;
    wire(3,0,-27).dx(-21); wire(3,0,-27).dx(2);! refresh;
    wire(3,0,-21).dx(-21); wire(3,0,-21).dx(2);! read ;
    wire(3,0,gy+7+6.5).dx(-21); wire(3,0,gy+7+6.5).dx(2);! write ;
    wire(3,0,gy+7+6.5+7).dx(-21); wire(3,0,gy+7+6.5+7).dx(2);!gnd ;
    wire(3,0,gy+7+6.5+14).dx(-21); wire(3,0,gy+7+6.5+14).dx(2);
    !reset ;
    rb(-4,gy+7+7);! write contact;
    layer(green);
    wire(2,0,gy+7+6.5+7).dx(-15);! green short line;
    gb(-15,gy+7+6.5+7);! green contact;
    wire(2,0,0).dy(35+14); wire(2,0,0).dy(-30);! control line;
    wire(2,-6,0).dy(13).dx(1).dy(5).dx(-7);
    wire(2,-6,0).dy(-12).dx(-6);
    gb(-12,gy);
```

```
    wire(2,-12,0).dy(gy);wire(2,-12,0).dy(-17).dx(+12);
    gb(-6,0); ! pull up to power;
    wire(2,-18,0).dy(gy+7).dx(+18);
    be(-13,gy+7);
    wire(2,-18,0).dy(-17-5).dx(8).dy(5);
    bs(-6,12);bn(-6,-12);
    layer(red);!pull ups;
    wire(2,-10,gy+7+6.5+14).dy(-10);rb(-10,gy+7+6.5+14);!reset ;
    wire(2,-8,gy+6).dy(-8);
    wire(6,-6,-7).dy(2);wire(6,-6,+7).dy(-2);
    wire(2,-6,10).dx(-9);! input trans;
    wire(2,-6,-21).dy(6);
    rb(-6,-20.5);
    rb(-13,-27.5);
    wire(2,-13,-27).dy(7);
    wire(2,-4,gy+7+7).dy(-7-3);! write contact;
    layer(implant);
    wire(5,-6,-7).dy(2);wire(5,-6,+7).dy(-2);
END;
enddef;

define("subendcell");
!-------------------------------------------------------------------------------;
BEGIN
    INTEGER x,gy;
    gy:=18;!ground wire;
    ! the origin is at the center or the input to pla;
    layer(blue);
    wire(4,0,0).dx(-21); wire(3,0,0).dx(2);! vdd ;
    wire(3,0,-27).dx(-21); wire(3,0,-27).dx(2);! refresh;
    wire(3,0,-21).dx(-21); wire(3,0,-21).dx(2);! read ;
    wire(3,0,gy+7+6.5).dx(-21); wire(3,0,gy+7+6.5).dx(2);! write ;
    wire(3,0,gy+7+6.5+7).dx(-21); wire(3,0,gy+7+6.5+7).dx(2);!gnd ;
    wire(3,0,gy+7+6.5+14).dx(-21); wire(3,0,gy+7+6.5+14).dx(2);
    !reset ;
    layer(green);
    wire(2,0,gy+7+6.5+7).dx(-11);! green short line;
    gb(-11,gy+7+6.5+7);! green contact;
    wire(2,0,0).dy(35+14); wire(2,0,0).dy(-30);! control line;
    wire(2,-6,0).dy(-12).dx(-6);
    wire(2,-12,-12).dy(-5).dx(+12);
    gb(-6,0); ! pull up to power;
    bn(-6,-12);
    layer(red);!pull ups;
    wire(2,-6,gy+7+6.5+14).dy(-10);rb(-6,gy+7+6.5+14);!reset ;
    wire(6,-6,-7).dy(2);
    wire(2,-6,-21).dy(6);
    rb(-6,-20.5);
    layer(implant);
    wire(5,-6,-7).dy(2);
END;
enddef;

rom:-NEW pla(7,74,34,0,"ucode",IF romguts THEN 10 ELSE 2);
bluebuzsep:=4+3;
leftpnt:-NEW point(0,0);
rightpnt:-NEW point(0,0);


define("rom");
```

```
!-------------------------------------------------------------------------------;
edgerompnt:-NEW point(0,-15.5);
drawmy(rom.planame,0,0);
enddef;

define("subregister");
!-------------------------------------------------------------------------------;
x:=rom.plainput(7).x+14;y:=rom.plainput(7).y+27+22;
adjustpntx(x);adjustpnty(y);
subtoppnt:-NEW vector;subbottompnt:-NEW vector;
! generation of the subroutine cell;
subreadpnt:-NEW point(x+2,y-21);
subwritepnt:-NEW point(x+2,y+18+7+6.5);
v:- NEW vector;
FOR i:=7 STEP -1 UNTIL 1 DO
BEGIN
    subbottompnt.append(NEW point(x,y-30));
    v.append(NEW point(rom.plainput(i).x,rom.plainput(i).y));
    subtoppnt.append(NEW point(x,y+35+14));
    IF i=7 THEN
    BEGIN
        draw("subendcell",x,y);
        x:=x-17;
    END
    ELSE
    BEGIN
        draw("subprimcell",x,y);
        x:=x-23;
    END;
END;
adjustpntx(x);adjustpnty(y);
subvddpnt:-NEW point(x,y);
subgndtpnt:-NEW point(x,y+18+7+6.5+7);
subgndbpnt:-NEW point(x,y+18);
subph2pnt:-NEW point(x,y-27);
subenapnt:-NEW point(x,y-21);
subresetpnt:-NEW point(x,y+18+7+6.5+14);

! lets connect up the sub to rom;
layer(green);
riverfrank(v,subbottompnt);
v:-NONE;

! generation of the top bus interface;
! subtoppnt is changed to point to the top points!;
layer(green);
j:=8;! height from top of sub for now;
FOR i:=1 STEP 1 UNTIL 7 DO
BEGIN
    pnt:-subtoppnt.val(i) QUA point;
    wire(2,pnt.x,pnt.y).dy(j);
    pnt.y:=pnt.y+j;
    gb(pnt.x,pnt.y);
    j:=j+bluebuzsep;
    adjustpntx(pnt.x);adjustpnty(pnt.y);
END;
adjustpntx(x);adjustpnty(pnt.y);
topy:=pnt.y;
! generate phase1 cut off for the write into sub;
layer(red);
```

```
pnt:=rom.phi1;
pnt.x:=rom.plainput(7).x+8;pnt.y:=-pnt.y;
wire(2,pnt.x,pnt.y).y(rom.plaoutput(1).y+6)
.x(rom.plaoutput(1).x+3);
adjustpntx(x);adjustpnty(y);

enddef;

adjustpntx(x);adjustpnty(y);
!-------------------------------------------------------------------------;
!      D   A   V   E   S       P     O   I   N   T   S                    ;
!-------------------------------------------------------------------------;
! now lets create the zoom;
allportid:-NEW vector; ! for the translater;
pascpnt:-NEW vector;
pascpnt.append(NEW point(1216,-651));!_WRITE_TO_LOWER (BUF) (PLSR) #1;
pascpnt.append(NEW point(1192,-651));
!I.R.:_WRITE_TO_UPPER (BUF) (PLSR) #1;
pascpnt.append(NEW point(1168,-651));
!I.R.:_READ_FROM_UPPER (BUF) (PLSR) #1;
pascpnt.append(NEW point(1120,-651));
!ALU_OUT:_WRITE_TO_UPPER (BUF) (PLSR) #1;
pascpnt.append(NEW point(1024,-651));!CIN (BUF) (PLSR) #1;
pascpnt.append(NEW point(856,-651));!KFF (BUF) (PLSR) #1;
pascpnt.append(NEW point(832,-651));!KFT (BUF) (PLSR) #1;
pascpnt.append(NEW point(808,-651));!KTF (BUF) (PLSR) #1;
pascpnt.append(NEW point(784,-651));!KTT (BUF) (PLSR) #1;
pascpnt.append(NEW point(760,-651));!PFF (BUF) (PLSR) #1;
pascpnt.append(NEW point(736,-651));!PFT (BUF) (PLSR) #1;
pascpnt.append(NEW point(712,-651));!PTF (BUF) (PLSR) #1;
pascpnt.append(NEW point(688,-651));!PTT (BUF) (PLSR) #1;
pascpnt.append(NEW point(616,-651));
!REGISTERS:_WRITE_TO_LOWER (BUF) (PLSR) #6;
pascpnt.append(NEW point(592,-651));
!REGISTERS:_WRITE_TO_UPPER (BUF) (PLSR) #6;
pascpnt.append(NEW point(544,-651));
!REGISTERS:_READ_FROM_UPPER (BUF) (PLSR) #6;
pascpnt.append(NEW point(520,-651));
!REGISTERS:_WRITE_TO_LOWER (BUF) (PLSR) #5;
pascpnt.append(NEW point(496,-651));
!REGISTERS:_WRITE_TO_UPPER (BUF) (PLSR) #5;
pascpnt.append(NEW point(472,-651));
!REGISTERS:_READ_FROM_UPPER (BUF) (PLSR) #5;
pascpnt.append(NEW point(448,-651));
!REGISTERS:_WRITE_TO_LOWER (BUF) (PLSR) #4;
pascpnt.append(NEW point(424,-651));
!REGISTERS:_WRITE_TO_UPPER (BUF) (PLSR) #4;
pascpnt.append(NEW point(400,-651));
!REGISTERS:_READ_FROM_UPPER (BUF) (PLSR) #4;
pascpnt.append(NEW point(376,-651));
!REGISTERS:_WRITE_TO_LOWER (BUF) (PLSR) #3;
pascpnt.append(NEW point(352,-651));
!REGISTERS:_WRITE_TO_UPPER (BUF) (PLSR) #3;
pascpnt.append(NEW point(328,-651));
!REGISTERS:_READ_FROM_UPPER (BUF) (PLSR) #3;
pascpnt.append(NEW point(304,-651));
!REGISTERS:_WRITE_TO_LOWER (BUF) (PLSR) #2;
pascpnt.append(NEW point(280,-651));
!REGISTERS:_WRITE_TO_UPPER (BUF) (PLSR) #2;
pascpnt.append(NEW point(256,-651));
```

```
!REGISTERS:_READ_FROM_UPPER (BUF) (PLSR) #2;
pascpnt.append(NEW point(232,-651));
!REGISTERS:_WRITE_TO_LOWER (BUF) (PLSR) #1;
pascpnt.append(NEW point(208,-651));
!REGISTERS:_WRITE_TO_UPPER (BUF) (PLSR) #1;
pascpnt.append(NEW point(184,-651));
!REGISTERS:_READ_FROM_UPPER (BUF) (PLSR) #1;
pascpnt.append(NEW point(160,-651));
!PORT:_READ_FROM_UPPER (BUF) (PLSR) #1;
pascpnt.append(NEW point(136,-651));
!PORT:_WRITE_TO_UPPER (BUF) (PLSR) #1;
pascpnt.append(NEW point(112,-651));
!PORT:_WRITE_TO_LOWER (BUF) (PLSR) #1;

portload:-NEW point(88,-651);!PORT, LOAD REG (BUF) (PLSR) #1;
allportid.append(portload);
portenable:-NEW point(64,-651);!PORT, DRIVE ENABLE (BUF) (PLSR) #1;
allportid.append(portenable);
zeropnt:-NEW point(1048,-651);!MSB (BUF) (PLSR) #1;
allportid.append(zeropnt);
msbpnt:-NEW point(1072,-651);!ZERO (BUF) (PLSR) #1;
allportid.append(msbpnt);
plsrin1:-NEW point(30,-651);!PLSR IN #0;
allportid.append(plsrin1);
plsrout1:-NEW point(1250,-651);!PLSR OUT #0;
allportid.append(plsrout1);
lsb:-NEW vector;
lsb.append(NEW point (1053,971));
lsb.append(NEW point(1169,971+18));
lsb.append(NEW point(1169,1079+18));
j:=lsb.length;
FOR i:=1 STEP 1 UNTIL j DO
allportid.append(lsb.val(i));
phidavepnt:-NEW point(-16+14,-651-2);
allportid.append(phidavepnt);

j:=pascpnt.length;
FOR i:=1 STEP 1 UNTIL j DO
allportid.append(pascpnt.val(i));

portpd:-NEW vector;
portpd.append(NEW point(0,24));!PORT, TO PAD #1;
portpd.append(NEW point(0,168));!PORT, TO PAD #2;
portpd.append(NEW point(0,312));!PORT, TO PAD #3;
portpd.append(NEW point(0,456));!PORT, TO PAD #4;
portpd.append(NEW point(0,600));!PORT, TO PAD #5;
portpd.append(NEW point(0,744));!PORT, TO PAD #6;
portpd.append(NEW point(0,888));!PORT, TO PAD #7;
portpd.append(NEW point(0,1032));!PORT, TO PAD #8;
j:=portpd.length;
FOR i:=1 STEP 1 UNTIL j DO
allportid.append(portpd.val(i));

irportpd:-NEW vector;
irportpd.append(NEW point(1271,1050));
irportpd.append(NEW point(1271,906));
irportpd.append(NEW point(1271,762));
irportpd.append(NEW point(1271,618));
j:=irportpd.length;
FOR i:=1 STEP 1 UNTIL j DO
```

```
allportid.append(irportpd.val(i));


j:=allportid.length;
FOR i:=1 STEP 1 UNTIL j DO
BEGIN
    pnt:-allportid.val(i) QUA point;
    x:=pnt.x/2;pnt.x:=pnt.y/2;
    pnt.y:=x;
    ! trans late;
    pnt.x:=pnt.x+(655/2);
    pnt.y:=pnt.y-(1280/2);   ! off one;
    !                ^ y translate value;
END;

v:-NEW vector;
FOR i:=1 STEP 1 UNTIL 34 DO
v.append(NEW point(pascpnt.val(i) QUA point.x,
-pascpnt.val(i) QUA point.y));
zoom:-NEW plaand(14,v,"zcode",IF zoomguts THEN 10 ELSE 2);
v:-NONE;

define("topbuz");
!-------------------------------------------------------------------;
! generate write line or hl line;
layer(green);
wire(2,rom.plaoutput(1).x,rom.plaoutput(1).y).y(subwritepnt.y);
gb(rom.plaoutput(1).x,subwritepnt.y);
layer(blue);
wire(3,rom.plaoutput(1).x,subwritepnt.y).x(subwritepnt.x);

! generates na-source lines;
layer(red); ! enable read on next address;
wire(2,rom.plaoutput(9).x,rom.plaoutput(9).y).y(subreadpnt.y-4)
.x(rom.plaoutput(2).x-3);
layer(green); ! read line on sub-register;
wire(2,rom.plaoutput(10).x,rom.plaoutput(10).y).y(subreadpnt.y);
gb(rom.plaoutput(10).x,subreadpnt.y);
layer(blue);
wire(3,subreadpnt.x,subreadpnt.y).x(rom.plaoutput(10).x);
! enable read on IR register;
layer(blue);
y:=subtoppnt.val(1)QUA point.y-bluebuzsep;
nasource:-NEW point(rom.plaoutput(34).x+5,y);
wire(3,rom.plaoutput(11).x,rom.plaoutput(11).y)
.y(nasource.y).x(nasource.x);
! generator the poly control for the top three to ground;
layer(red);
rb(rom.plaoutput(11).x,subtoppnt.val(1) QUA point.y-bluebuzsep);
wire(2,rom.plaoutput(11).x,subtoppnt.val(1) QUA point.y-bluebuzsep)
.y(subtoppnt.val(7) QUA point.y+3);

! generate condition codes;
! based on above defined y!!!!;
ccpnt:-NEW vector;x:=rom.plaoutput(34).x+5;
FOR i:=13 STEP 1 UNTIL 16 DO
BEGIN
    layer(blue);
    y:=y-bluebuzsep;
    wire(3,rom.plaoutput(i).x,rom.plaoutput(i).y).
```

```
    y(y).x(x);
    ccpnt.append(NEW point(x,y));
END;
layer(red);! disables line 0 from nextaddress;
wire(2,rom.plaoutput(12).x,rom.plaoutput(12).y)
.y(subreadpnt.y+6).x(rom.plaoutput(8).x-2);

! generation of top big blue buses from subtoppnt;
layer(blue);
x:=rom.plaoutput(34).x+5;
FOR i:=1 STEP 1 UNTIL 4 DO
BEGIN
    pnt:-subtoppnt.val(i) QUA point;
    wire(3,pnt.x,pnt.y).x(x);
    pnt.x:=x;
END;
x:=rom.plaoutput(9).x;
adjustpntx(x);adjustpnty(y);
FOR i:=5 STEP 1 UNTIL 7 DO
BEGIN
    pnt:-subtoppnt.val(i) QUA point;
    wire(3,pnt.x,pnt.y).x(x);
    gb(x,pnt.y);
END;
! greeny interface for the buz;
layer(green);
FOR i:=5 STEP 1 UNTIL 7 DO
BEGIN
    pnt:-subtoppnt.val(i) QUA point;
    wire(2,x,pnt.y).x(rom.plaoutput(13).x);
    gb(rom.plaoutput(13).x,pnt.y);
END;
! blue ground buz;
layer(blue);
topgndpnt:- NEW point(rom.plaoutput(13).x,subtoppnt.val(7)QUA
point.y+3);
wire(3,topgndpnt.x,subtoppnt.val(5)QUA point.y).
y(topgndpnt.y);

! generation of greeny interface from the big rom;
layer(green);
j:=1+7;
FOR i:=1 STEP 1 UNTIL 7 DO
BEGIN
    pnt :- subtoppnt.val(i) QUA point;
    wire(2,rom.plaoutput(j).x,rom.plaoutput(j).y).y(pnt.y);
    gb(rom.plaoutput(j).x,pnt.y);
    j:=j-1;
END;
adjustpntx(x);adjustpnty(y);

! this is the set of the x that determines the sep of the plaand;
x:=rom.plawidth-11;
! lets make the ground buz on the rom bigger - OK!!!!;
layer(blue);
wire(8,x,-4).y(-rom.plaheight);
romgndpnt:-NEW point(x,-rom.plaheight);
x:=x+9;

! buz control lines to bottom of chip in poly;
```

```
layer(blue);
!x:=x+2+3+1.5;
y:=rom.plaoutput(34).y+6;
buscntlpnt:-NEW vector;
FOR i:=34 STEP -1 UNTIL 32 DO
BEGIN
    wire(3,rom.plaoutput(i).x,rom.plaoutput(i).y).y(y).x(x).
    y(-rom.plaheight+18);
    buscntlpnt.append(NEW point(x,-rom.plaheight+18));
    x:=x+7;y:=y+6;
END;

! set ready line point;
rdypnt:-NEW point(x,-rom.plaheight+18);
pnt:-ccpnt.val(4) QUA point;
wire(3,rdypnt.x,rdypnt.y).y(pnt.y-bluebuzsep);
gb(rdypnt.x,pnt.y-bluebuzsep);
layer(green);
wire(2,rdypnt.x,pnt.y-bluebuzsep).
dx(-8).y(subtoppnt.val(1) QUA point.y);
gb(rdypnt.x-8,subtoppnt.val(1) QUA point.y);
rdytpnt:-NEW point(rdypnt.x,pnt.y-bluebuzsep);
layer(red);
rb(pnt.x,pnt.y);
wire(2,pnt.x,pnt.y).x(rdypnt.x-8+3);
x:=x+7;

! phase2 output to the bottom of the chip;
layer(red);
pnt:-rom.phi2;
wire(2,pnt.x,pnt.y).x(x);
rb(x,pnt.y);
layer(blue);wire(3,x,pnt.y).y(-rom.plaheight+24);
ph2romoutpnt:-NEW point(x,-rom.plaheight+24);


zoomx:=(x)+3+15.5;! set up the old zoom;
!              ^ clearance ;
!                ^ center of decoded;

! now lets lace the rom outputs to zoom inputs !!!!!;
BEGIN
    INTEGER bignurd;
    INTEGER i,j;
    i:=31;
    bignurd:=rom.plaoutput(34).y+6;
    FOR j:=1 STEP 1 UNTIL 14 DO
    BEGIN
        IF mod(j,2)=1 THEN
        BEGIN
            layer(red);
            wire(2,rom.plaoutput(i).x,rom.plaoutput(i).y)
            .y(bignurd).x(zoom.plainput(j).x+zoomx)
            .y(zoom.plainput(j).y);
            bignurd:=bignurd+3;
        END ELSE
        BEGIN
            layer(green);
            wire(2,rom.plaoutput(i).x,rom.plaoutput(i).y)
            .y(bignurd).x(zoom.plainput(j).x+zoomx)
```

```
            .y(zoom.plainput(j).y);
            bignurd:=bignurd+3;
        END;
        i:=i-1;
    END;
END;

! now lets translates zoom cor;
FOR i:=1 STEP 1 UNTIL 34 DO
BEGIN
    zoom.output(i).x:=zoom.output(i).x+zoomx;
    zoom.output(i).y:=-zoom.output(i).y;
END;

! now the subtop pnt is adjusted to be only the 4 points;
v:-NEW vector;
FOR i:=1 STEP 1 UNTIL 4 DO v.append(subtoppnt.val(i));
subtoppnt:-v;

enddef;

define("zoom");
!-------------------------------------------------------------------------;
edgezoompnt:-NEW point(zoomx-15.5,0);
drawmy(zoom.planame,zoomx,0);
gregpnt:-NEW point(zoomx,-zoom.plaheight);
enddef;

! determine  daves stuff position !!!;
zoomx:=zoomx+zoom.plawidth+35+3+8+0;
!                              ^ five metal lines;
!                                 ^ clearance;
!                                   ^ bonus;
!                                      ^ gnd;

! trans all ports by zoom;
j:=allportid.length;
FOR i:=1 STEP 1 UNTIL j DO
BEGIN
    ! trans late x by zoom;
    pnt:-allportid.val(i) QUA point;
    pnt.x:=pnt.x+zoomx;
END;

! -----------------------------------------------------------------------;
IF NOT breadcif THEN !readcif;
BEGIN
    define("pascalchip");
    layer(green);
    !box(-65/2,-655/2,1340/2,1108/2);
    ! a box of the data path;
    enddef;
END;
davesize:-NEW point(1108/2+655/2,1340/2+65/2);

define("daver");
!-------------------------------------------------------------------------;
drawrot("pascalchip",0,0,0,-1);
enddef;
```

```
define("dave");
! ------------------------------------------------------------------;
drawmy("daver",(655/2)+zoomx,-1280/2);! took off 100;
edgedavepnt:-NEW point(zoomx,30);
enddef;

define("congeorge");
!-------------------------------------------------------------------;
BEGIN
    REAL   basex,basey,z;
    REF(point)   cc3pnt;
    ! top ground buz from big dave to the old zoom'0;
    layer(blue);
    wire(20,edgedavepnt.x,edgedavepnt.y-10).x(edgezoompnt.x+
    zoom.plawidth-12).y(edgezoompnt.y-40);

    ! set bases for up lines;
    basey:=edgedavepnt.y-20-3-1.5;
    zorzk:-new vector;
    basex:=zeropnt.x-3-2;

    ! clock line ;
    layer(green);
    wire(3,plsrout1.x+2,plsrout1.y+6).
    x(basex).
    y(basey);
    gb(basex,basey);
    zorzk.append(new point(basex,basey));
    basex:=basex-7;

    ! plsrout1 ----------------------;
    layer(red);
    wire(2,plsrout1.x+2,plsrout1.y).
    x(basex).z(blue).
    y(basey);
    layer(green);
    rb(basex,basey);
    zorzk.append(new point(basex,basey));
    basex:=basex-7;


    ! condition code set;
    cc3pnt:-ccpnt.val(3) QUA point;
    ! msb;
    layer(blue);
    rb(msbpnt.x,msbpnt.y);
    pnt:-subtoppnt.val(1) QUA point;
    wire(3,msbpnt.x,msbpnt.y).
    x(basex).
    y(basey);
    layer(green);
    gb(basex,basey);
    zorzk.append(new point(basex,basey));
    wire(2,basex,basey).
    y(cc3pnt.y-15).x(basex+12).
    y(pnt.y);
    gb(basex+12,pnt.y);
    layer(blue);
    wire(3,cc3pnt.x,cc3pnt.y).x(basex+12-5);
    rb(basex+12-5,cc3pnt.y);
```

```
layer(red);
wire(2,basex+12-5,cc3pnt.y).x(basex+12-5+8);
basex:=basex-7;

!zero;
layer(blue);
rb(zeropnt.x,zeropnt.y);
pnt:=subtoppnt.val(1) QUA point;
wire(3,zeropnt.x,zeropnt.y).
x(basex).
y(basey);
layer(green);
gb(basex,basey);
zorzk.append(new point(basex,basey));
wire(2,basex,basey).
y(pnt.y);
gb(basex,pnt.y);
cc3pnt:=ccpnt.val(2) QUA point;
layer(blue);
wire(3,cc3pnt.x,cc3pnt.y).x(basex-5);
rb(basex-5,cc3pnt.y);
layer(red);
wire(2,basex-3,cc3pnt.y).x(basex+3);
basex:=basex-7;

! plsrin1 ----------------------;
layer(blue);
rb(plsrin1.x,plsrin1.y+1);
wire(3,plsrin1.x,plsrin1.y).
x(basex).
y(basey);
layer(green);
gb(basex,basey);
zorzk.append(new point(basex,basey));
basex:=basex-7;

! plsrs1 ----------;
gb(basex,basey);
zorzk.append(new point(basex,basey));
layer(blue);
wire(3,basex,basey).
y(edgedavepnt.y-davesize.y+30+8).dx(12).
z(green).
x(edgedavepnt.x+5);
basex:=basex-7;

! phi1 ----------;
! adjustment of ph1 for design rules;
phidavepnt.y:=phidavepnt.y-2;
gb(basex,basey);
zorzk.append(new point(basex,basey));
layer(blue);
wire(3,basex,basey).
y(phidavepnt.y).x(edgedavepnt.x-5).
z(green).
x(phidavepnt.x);
phidavepnt.x:=edgedavepnt.x-5;
gb(phidavepnt.x,phidavepnt.y);

! top connections to the IR;
```

```
    layer(red);
    j:=pascpnt.length;
    FOR i:=1 STEP 1 UNTIL j DO
    BEGIN
        pnt:-pascpnt.val(i) QUA point;
        pnt.y:=pnt.y-3;
    END;
    riverfredy(2,5,zoom.outzip,pascpnt);
    v:-NEW vector;
    x:=irportpd.val(4) QUA point.x;
    y:=subtoppnt.val(1) QUA point.y-10;
    FOR i:=4 STEP -1 UNTIL 1 DO
    BEGIN
        pnt:-subtoppnt.val(i) QUA point;
        layer(blue);
        IF i=1 THEN topline:-NEW point(x,pnt.y);
        wire(3,pnt.x,pnt.y).x(x);
        gb(x,pnt.y);
        layer(green);
        wire(2,x,pnt.y).y(irportpd.val(i) QUA point.y+15+z).
        x(irportpd.val(i) QUA point.x-4).
        y(irportpd.val(i) QUA point.y);
        z:=z+7;
        x:=x+bluebuzsep;
    END;
    pnt:-irportpd.val(4) QUA point;
    layer(blue);
    wire(3,nasource.x,nasource.y).x(pnt.x-8);
    rb(pnt.x-8,nasource.y);
    layer(red);
    wire(2,pnt.x-8,nasource.y).x(x-4);
END;
enddef;

greg:-NEW pla(9,12,8,4,"gcode",IF gregguts THEN 10 ELSE 2);
define("greg");
!-----------------------------------------------------------------------;
BEGIN
    gregpnt.y:=gregpnt.y-14;
    gregpnt.x:=gregpnt.x+20;
    drawrot(greg.planame,gregpnt.x,gregpnt.y,0,-1);
    j:=3;
    layer(blue);
    FOR i:=1 STEP 1 UNTIL 3 DO
    BEGIN
        pnt:-buscntlpnt.val(j) QUA point;j:=j-1;
        wire(3,pnt.x,pnt.y).y(gregpnt.y-(i+1)*14).
        x(gregpnt.x-2);
        gb(gregpnt.x-2,gregpnt.y-(i+1)*14);
    END;
    wire(3,rdypnt.x,rdypnt.y).y(gregpnt.y-14).
    x(gregpnt.x-2);
    gb(gregpnt.x-2,gregpnt.y-14);
    pnt:-greg.phi1;pnt.y:=pnt.y+1;
    x:=pnt.x;pnt.x:=pnt.y;pnt.y:=x;
    wire(3,ph2romoutpnt.x,ph2romoutpnt.y).
    y(gregpnt.y-pnt.y).
    x(gregpnt.x+pnt.x);
    rb(gregpnt.x+pnt.x,gregpnt.y-pnt.y);
```

```
    ! now for the power to greg and zoom;
    layer(blue);
    wire(6,edgezoompnt.x+3+4,gregpnt.y+15.5-5),
    x(gregpnt.x+greg.plaheight-8),
    y(edgedavepnt.y-davesize.y-5);

END;
enddef;


define("elbottompads");
!-------------------------------------------------------------------------;
BEGIN
    INTEGER xgndst,xgnd,xena,yena,basey;
    INTEGER z,u;
    REF(vector) p;
    x:=portpd.val(8) QUA point.x-3;
    xgnd:=x:=x-54-32;
    v:-NEW vector;
    basey:=y:=portpd.val(1) QUA point.y-60;
    FOR i:=1 STEP 1 UNTIL 8 DO
    BEGIN
        drawrot("padtristate",x,y,0,-1);
        layer(green);
        v.append(NEW point(x+24,y));
        wire(2,x+24-6,y).x(x+2);
        be(x+24-6,y-2);
        gb(x+3,y-2);
        IF i=1 THEN BEGIN xena:=x+100;yena:=y+5;END;
        layer(red);
        wire(2,x+100,y).y(y+5);
        rb(x+100,y+5);
        wire(2,x+2+8,y-3).y(y+5+7);
        rb(x+2+8,y+5+7);
        j:=j-1;
        x:=x-106;
    END;
    xgndst:=x+106;
    layer(blue);
    wire(3,xena,yena).x(portenable.x);
    enableport:-NEW point(portenable.x,yena);
    wire(3,xena,yena+7).x(portenable.x);
    loadport:-NEW point(portenable.x,yena+7);
    layer(red);
    riverfredx(5,15,v,portpd);
    v:-NONE;
    x:=-20;
    ! power pad;
    drawrot("ayatollah",x,y-170,-1,0);
    p:-NEW vector;
    z:=y-170+145+(4*7);u:=-20+4*106+3+6+4+3+2;
    FOR i:=1 STEP 1 UNTIL 4 DO
    BEGIN
        drawrot("padout",x+106,y-170,-1,0);
        layer(red);
        wire(2,x-106/2+106,y-170+145
        ).y(z).x(u);p.append(NEW point(u,z));
        rb(u,z);
        z:=z-7;
        x:=x+106;
```

```
END;
v:-NEW vector;
! pads for connection to greg;
pnt:-NEW point(gregpnt.x+greg.plainput(6).y,
gregpnt.y-greg.plainput(6).x);
pnt.x:=pnt.x-2;gb(pnt.x,pnt.y);
v.append(pnt);
pnt:-NEW point(gregpnt.x+greg.plainput(7).y,
gregpnt.y-greg.plainput(7).x);
pnt.x:=pnt.x-2;gb(pnt.x,pnt.y);
v.append(pnt);
pnt:-NEW point(gregpnt.x+greg.plaoutput(5).y,
gregpnt.y-greg.plaoutput(5).x);
v.append(pnt);
pnt:-NEW point(gregpnt.x+greg.plaoutput(6).y,
gregpnt.y-greg.plaoutput(6).x);
v.append(pnt);
layer(blue);
riverfredy(3,7,p,v);
p:-NONE;v:-NONE;

layer(metal);
! power;
wire(20,edgerompnt.x+24-20-106+10,y-170-10+7),
x(edgedavepnt.x+davesize.x-10),
y(edgedavepnt.y-davesize.y);
wire(20,edgerompnt.x+24-20-106+10,y-170-10+7),
y(topy+10+145+2),
x(edgedavepnt.x+davesize.x-10);

!ground;
wire(20,xgnd-10-12+106,y-170+106-10),
x(xgndst-10);
wire(20,-20+10+12,y-170+106-10),
x(-20+(4*106)+3),
y(gregpnt.y-10-greg.plawidth+15.5-2),
x(xgndst-10),
y(y-170+106-10);
! ground up to zoom;
wire(20,edgezoompnt.x+zoom.plawidth-12,
gregpnt.y-10-greg.plawidth+15.5-2+5),
y(edgezoompnt.y-zoom.plaheight+5);

! power from big dave over to the greg and to ZOOOOOOoooooommmmm!;
y:=edgedavepnt.y-davesize.y+10;
wire(20,edgedavepnt.x,y),
x(edgezoompnt.x+zoom.plawidth-12+20+3);
layer(green);
x:=edgezoompnt.x+zoom.plawidth-12+20+3-10-6;
wire(20,x,y),
dx(-15);
layer(blue);
box(gregpnt.x+greg.plaheight-8,y-10,x-14-7,y+10);
x:=x+8;y:=y+8;
FOR i:=1 STEP 1 UNTIL 5 DO
BEGIN
    gb(x,y);gb(x-31,y);y:=y-4;
END;

! ground to rom a dum;
```

```
    layer(blue);
    wire(8,romgndpnt.x,romgndpnt.y).
    y(basey+4+2+3).
    x(-20+(4*106)+3+6).
    y(basey-170+106-10);

END;
enddef;

define("caltech");
begin
text t;
t:=copy("CALTECH 1979");
layer(blue);
alpha(t,0,0,5);
end;
enddef;

define("eltoppads");
!-------------------------------------------------------------------------------;
BEGIN
    ref(point) p;
    text t;
    y:=topy+10+145;x:=edgerompnt.x;
    x:=edgerompnt.x+24-20-106+20+4;
    threetops:-NEW vector;
    FOR i:=1 STEP 1 UNTIL 3 DO
    BEGIN
        layer(green);
        pnt:-NEW point(x+98,y-110-(i*7));
        wire(2,pnt.x-2,y-102).y(pnt.y);
        gb(pnt.x,pnt.y);
        threetops.append(pnt);
        draw("padin",x,y);
        x:=x+106;
    END;

    ! ready contact connection;
    x:=rdytpnt.x;
    draw("padin",x-96,y);
    layer(green);
    wire(2,rdytpnt.x,rdytpnt.y).y(y-104);

    ! phase connections;
    ! phase   1;
    layer(blue);
    pnt:-threetops.val(2) qua point;
    wire(3,pnt.x,pnt.y).x(edgezoompnt.x+10);
    gb(edgezoompnt.x+10,pnt.y);
    p:-zorzk.val(7) qua point;
    layer(green);
    wire(2,edgezoompnt.x+10,pnt.y).
    y(edgedavepnt.y+30).
    x(p.x).
    y(p.y);

    ! phase 2;
    pnt:-threetops.val(3) qua point;
    p:-zorzk.val(7) qua point;
    layer(blue);
```

```
wire(3,pnt.x,pnt.y).
y(y-145-20).
x(edgedavepnt.x+46);
gb(edgedavepnt.x+46,y-145-20);
layer(green);
wire(2,edgedavepnt.x+46,y-145-20).
y(zorzk.val(1) qua point.y+2);

! ph2 from bottom of rom a rum;
layer(blue);
wire(3,ph2romoutpnt.x,rom.phi2.y).
y(ccpnt.val(4) qua point.y-bluebuzsep);
rb(ph2romoutpnt.x+1,ccpnt.val(4) qua point.y-bluebuzsep);
layer(red);
wire(2,ph2romoutpnt.x,ccpnt.val(4) qua point.y-bluebuzsep).
y(y-145-20);
rb(ph2romoutpnt.x,y-145-20);


! plsr stuff;
x:=edgedavepnt.x;
x:=x-106;
draw("padin",x-96,y);
pnt:=zorzk.val(5) qua point;
layer(green);
wire(2,pnt.x,pnt.y).y(edgedavepnt.y+40).
x(x).y(y-102);
x:=x-106;

draw("padin",x-96,y);
pnt:=zorzk.val(6) qua point;
layer(green);
wire(2,pnt.x,pnt.y).y(edgedavepnt.y+35).
x(x).y(y-102);
x:=x-106;

pnt:=zorzk.val(2) qua point;
x:=pnt.x-106/2+15;
draw("padout",x,y);
layer(red);
wire(2,pnt.x,pnt.y).y(edgedavepnt.y+20).
x(pnt.x+15).y(y-145+2);
x:=x+106;

draw("padin",x,y);
pnt:=zorzk.val(1) qua point;
layer(green);
wire(2,pnt.x,pnt.y).y(edgedavepnt.y+12).
x(pnt.x+18).y(y-145-10).
x(x+96).y(y-102);
x:=x+110;

! put are names here;
layer(blue);
t:=copy(" G EFLAND");
alpha(t,x,y-62,5);
t:=copy(" R C MOSTELLER");
alpha(t,x,y-106-45-15,5);
t:=copy("STACK DATA ENGINE");
alpha(t,edgedavepnt.x+60,edgedavepnt.y+10,5);
```

```
    ! top ground stuff ;;
    layer(blue);
    box(edgedavepnt.x+davesize.x-100,y-106,
        edgedavepnt.x+davesize.x,
        y-106+30);
    draw("padblank",edgedavepnt.x+davesize.x-106,y);
    layer(blue);
    wire(20,edgedavepnt.x+davesize.x-10,edgedavepnt.y).
    y(y-106+10).
    x(edgerompnt.x+24-20-106+20+3+2+8.5);
    layer(green);
    wire(4,topgndpnt.x,topgndpnt.y).
    y(y-106+10);



END;
enddef;

define("leftbuz");
!-------------------------------------------------------------------------------;
BEGIN
    REAL x,y;
    layer(blue);
    !power for rom;
    wire(8,edgerompnt.x-11,edgerompnt.y).
    y(edgerompnt.y-rom.plaheight+4).
    x(edgerompnt.x+24-20-106+2);
    ! power for sub;
    wire(3,edgerompnt.x+24-20-106+2,subvddpnt.y).x(subvddpnt.x);

    ! ground for sub;
    wire(3,subgndbpnt.x,subgndbpnt.y).
    x(edgerompnt.x+24-20-106+20+3+2).
    y(topy+10+145-106+10);
    wire(3,subgndtpnt.x,subgndtpnt.y).
    x(edgerompnt.x+24-20-106+20+3+2);

    ! strap for load and read;
    x:=subgndtpnt.x;
    layer(blue);
    wire(3,x,subreadpnt.y).x(x-3);
    wire(3,x,subwritepnt.y).x(x-3);
    gb(x-3,subwritepnt.y);
    gb(x-3,subreadpnt.y);
    layer(green);
    x:=x-3;
    wire(2,x,subwritepnt.y).y(subreadpnt.y);
    gb(x,subgndbpnt.y);! contact to ground;

    ! phase 2 ;
    layer(red);
    y:=subresetpnt.y+3+4;
    x:=x-2-2;
    rb(subph2pnt.x,subph2pnt.y);
    wire(2,subph2pnt.x,subph2pnt.y).x(x).
    y(y);
    wire(2,x,subreadpnt.y+5).x(x+2+2+3);
    wire(2,x,subwritepnt.y-5).x(x+2+2+3);
```

```
    rb(x,y);

    ! phase1 to rom;
    x:=x-7;
    rb(x,y);
    layer(red);
    wire(2,x,y).y(-rom.phi1.y).x(rom.phi1.x);

    ! reset line;
    x:=x-7;
    layer(blue);
    wire(3,subresetpnt.x,subresetpnt.y).x(x).y(y);
    rb(x,y);
    layer(red);
    wire(2,x,y).y(gregpnt.y-greg.plainput(5).x+3).
    x(-20+4*106+3+6+4+3+2);
    rb(-20+4*106+3+6+4+3+2,gregpnt.y-greg.plainput(5).x+4);
    layer(blue);
    wire(3,-20+4*106+3+6+4+3+2,gregpnt.y-greg.plainput(5).x+4).
    x(gregpnt.x-2).
    y(gregpnt.y-greg.plainput(5).x);
    gb(gregpnt.x-2,gregpnt.y-greg.plainput(5).x);

    ! add the connections up to the three tops !!!!!;
    layer(blue);
    FOR i:=1 STEP 1 UNTIL 3 DO
    BEGIN
        pnt:-threetops.val(i) QUA point;
        wire(3,x,y).y(pnt.y).x(pnt.x);
        x:=x+7;
    END;

END;
enddef;

define("woopee");
BEGIN
    REAL wx,wy;
    wx:=edgezoompnt.x+zoom.plawidth-2+22;
    wy:=enableport.y-100;
    drawmx("noninvertingsinglesb",wx,wy);
    draw("noninvertingsinglesb",wx,wy);

    ! power ;;
    layer(blue);
    wire(4,wx+20,wy+16).
    y(wy+43);
    wire(4,wx-16,wy+43).
    y(edgedavepnt.y-davesize.y);
    ! ground;
    wire(6,wx,wy+32).
    dx(-20);

    ! one output;
    layer(green);
    wire(2,wx-08,wy+43).
    y(portload.y).
    x(portload.x);
    layer(blue);
    wire(3,loadport.x,loadport.y).
```

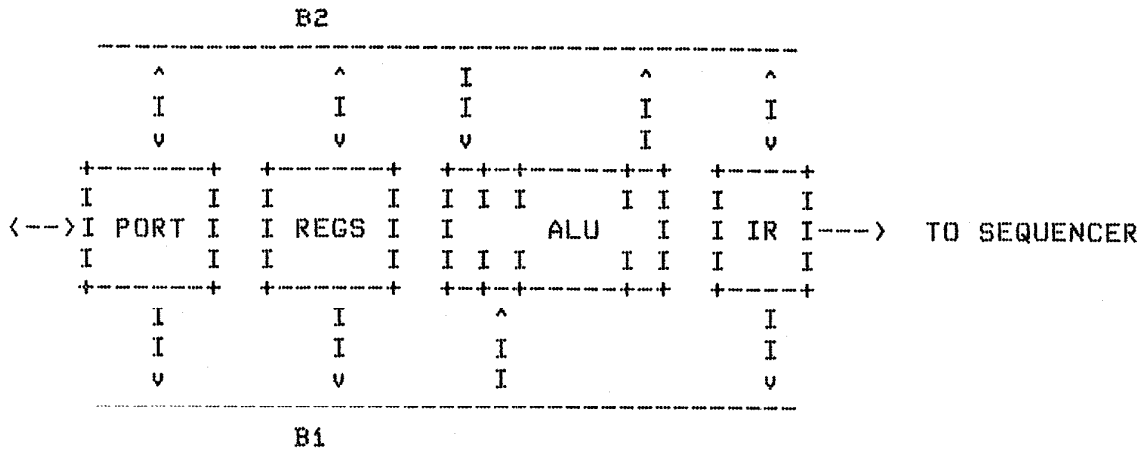
```
x(wx-08);
gb(wx-08,loadport.y);

! one output;
layer(green);
wire(2,wx+08,wy+43).
y(portenable.y).
x(portenable.x);
layer(blue);
wire(3,enableport.x,enableport.y).
x(wx+08);
gb(wx+08,enableport.y);

! one input;
layer(red);
wire(2,gregpnt.x+3,gregpnt.y-greg.plaoutput(7).x+1).
dx(-10).dy(-50).
x(wx+3).y(wy-9);

! two input;
wire(2,gregpnt.x+greg.plaheight-15.5-5
,gregpnt.y-greg.plaoutput(8).x+1).
dy(-12).
x(wx-3).y(wy-9);

! tick tock !!!;
layer(red);
wire(2,gregpnt.x+greg.phi2.y,gregpnt.y-greg.phi2.x).
dy(-27).
z(blue).x(phidavepnt.x).
z(green).y(phidavepnt.y);

! lsb ----------------------------------------;
layer(blue);
wire(3,lsb.val(1) QUA point.x,lsb.val(1) QUA point.y).
x(lsb.val(2) QUA point.x).
y(lsb.val(2) QUA point.y);
layer(red);
rb(lsb.val(2) QUA point.x,lsb.val(2) QUA point.y);
wire(2,lsb.val(2) QUA point.x,lsb.val(2) QUA point.y).
x(lsb.val(3) QUA point.x).
y(topline.y-5).
x(topline.x+30+52);
layer(green);
wire(2,topline.x+30,topline.y).x(topline.x);
drawrot("noninvertingsinglesb",topline.x+30+43,
topline.y-8,0,1);
! add cc connect;
layer(blue);
pnt:=ccpnt.val(1) QUA point;
wire(3,pnt.x,pnt.y).x(topline.x+20);
rb(topline.x+20,pnt.y);
layer(red);
wire(2,topline.x+20,pnt.y).y(topline.y+3);

! super smucks groung;
layer(blue);
wire(6,topline.x+30+43-32,topline.y-8).
y(edgedavepnt.y);
```

```
            !super smucks power;
            layer(blue);
            wire(3,topline.x+30+43-16,topline.y+19-8),
            x(topline.x+30);

            wire(10,topline.x+30+4,topline.y+19-8+5),
            y(topy+10+145-106+10-23);
            wire(10,topline.x+30+4,topy+10+145-106+10+23),
            y(topy+10+145);
            layer(green);
            wire(10,topline.x+30+4,topy+10+145-106+10-23+5),
            y(topy+10+145-106+10+23-5);
            x:=topline.x+30+4+1;
            FOR i:=1 STEP 1 UNTIL 2 DO
            BEGIN
                gb(x,topy+10+145-106+10-23+5+2);
                gb(x,topy+10+145-106+10+23-5-2);
            END;

        END;
        enddef;

        define("GregandRickys_chip");
        !-------------------------------------------------------------------------;
        drawrot("caltech",edgerompnt.x-25,0-rom.plaheight+180,0,1);
        draw("woopee",0,0);
        draw("rom",0,0);
        draw("congeorge",0,0);
        draw("zoom",0,0);
        draw("subregister",0,0);
        draw("topbuz",0,0);
        draw("dave",0,0);
        draw("elbottompads",0,0);
        draw("eltoppads",0,0);
        draw("greg",0,0);
        draw("leftbuz",0,0);
        enddef;

        draw("gregandrickys_chip",0,0);
        outtext(" the supposed bounding box -x1=");
        outint(leftpnt.x,5);outtext(",highy=");
        outint(leftpnt.y,5);outtext(",xr=");
        outint(rightpnt.x,5);outtext(",lowy=");
        outint(rightpnt.y,5);outimage;

    END of def281 block;


END of program;
```

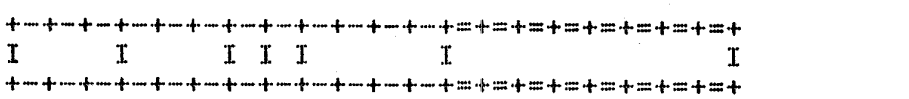
A P E N D I X ------- C

BRISTLE BLOCKS DESCRIPTION

```
"PASCAL.ICL:    Pascal machine datapath:"

"       First we will hack together the two port cells, since the fancy
        port routines have not yet been written.  We need an IO left and
        an output right producer.  We also need an alu with only 1 output
        register:"

DEFINE IO_LEFT(M:MEMORY_USPEC  A,B:UCODE_EQUATION  N:SC):
        INCLUDE(//[A;B;N;]
                AT_LEAST(Y1,40);
                AT_LEAST(Y2,39);
                AT_LEAST(Y3,14);
                AT_LEAST(Y4,14);
                AT_LEAST(Y6,16);
                ADD(DATAPATH_POWER,8);
                NEXT_GUY;
                INSERT('%%PORT_IO_RIGHT_COL'\CHECKFILE\MIR MIRY\AT 70\XY 0
                        \WITH_USPECS (A\NAMED N$$', LOAD REG';
                                        B\NAMED N$$', DRIVE ENABLE';
                                SAME\NAMED N$$', TO PAD';UNUSED;UNUSED));
                MOVE_ORIGIN(70);\\);
        FLIPPED_REGISTER(M\ADD_LINKS (3;6;12;14),N);
        STRETCHER;
ENDDEFN

DEFINE OUT_RIGHT(M:MEMORY_USPEC  N:SC):
        STRETCHER;
        REGISTER(M\ADD_LINKS (4),N);
        INCLUDE(//
                AT_LEAST(Y1,26);
                AT_LEAST(Y2,26);
                AT_LEAST(Y3,14);
                AT_LEAST(Y4,14);
                AT_LEAST(Y6,16);
                ADD(DATAPATH_POWER,5);
                NEXT_GUY;
                INSERT('%%PORT_OUT_RIGHT_COL'\CHECKFILE\WITH_USPECS (SAME\NAMED '
                MOVE_ORIGIN(20);\\);
ENDDEFN

DEFINE ALU(M1,M2,M3:MEMORY_USPEC  A:UCODE_EQUATIONS  S:SC):
        STRETCHER;
        REGISTER(M1\ADD_LINKS (7;8;11;12),S$$'_IN_A');
        REGISTER(M2\ADD_LINKS (3;4;9;10;11;12;13;14),S$$'_IN_B');
        INCLUDE(//[A;]
                AT_LEAST(Y1,50);
                AT_LEAST(Y2,50);
                AT_LEAST(Y3,10);
                AT_LEAST(Y4,10);
                AT_LEAST(Y6,26);
                AT_LEAST(Y5,24);
                ADD(DATAPATH_POWER,10);
                NEXT_GUY;
                INSERT('%%ALU'\CHECKFILE\WITH_USPECS A);
                MOVE_ORIGIN(391);\\);
        REGISTER(M3\ADD_LINKS (2),S$$'_OUT');
        STRETCHER;
ENDDEFN

" AND NOW, FOR THE CHIP:
```



MICROCODE:



CLOCK PHASE WE SAMPLE ON
I
V

2 B1ADR: Source register for b1 (always loads into alu)
- 0: IR
- 1-6: Reg
- 7: Port

2 B2ADR: Source or destination register for B2
- 0: IR
- 1-6: Reg
- 7: Port

2 B2RW: Direction of b2
- 0: Reading from register
- 1: Writing to register
- (note: it always writes into the alu also)

2 WRSRC: Source for b2
- 0: alu goes unto bus (if also B2RW is high)
- 1: Port goes unto bus (even if B2RW is LOW, so watch it!)

1 ALU: ALU operation:
- 0: illegal operation
- 1: A+1
- 2: A-B
- 3: A-B
- 4: A+B
- 5: A+B
- 6: A-1
- 7: A-1
- 8: A or B
- 9: A

```
        A:      A and B
        B:      A and B
        C:      A and -B
        D:      A and -B
        E:      0
        F:      -A (not A)

   WE ALSO PROVIDE THE ZERO, MSB, AND LSB FLAGS

SET(WIDTH,8);

CONTROL_WIDTH(20);

VAR B1ADR,B2ADR,B2RW,WRSRC,ALU,INTERNAL=FIELD;

B1ADR:= 3\BITS \STARTING_WITH_BIT 1;

B2ADR:= 3\BITS \STARTING_WITH_BIT 4;

B2RW:= INSTRUCTION\BIT 7;

WRSRC:= INSTRUCTION\BIT 8;

ALU:= 4\BITS \STARTING_WITH_BIT 9;

INTERNAL:= 8\BITS \STARTING_WITH_BIT 13;

"Lets name the bits so Dick can find them easily:"
        NAME(BIT(1),'B1ADR 1');
        NAME(BIT(2),'B1ADR 2');
        NAME(BIT(3),'B1ADR 3');
        NAME(BIT(4),'B2ADR 1');
        NAME(BIT(5),'B2ADR 2');
        NAME(BIT(6),'B2ADR 3');
        NAME(BIT(7),'B2RW');
        NAME(BIT(8),'WRSRC');
        NAME(BIT(9),'ALU 1');
        NAME(BIT(10),'ALU 2');
        NAME(BIT(11),'ALU 3');
        NAME(BIT(12),'ALU 4');


"Define the INTERNAL functions:  (unfortunately, we dont have a nice way to
 do that yet..."
        DEFINE DEFINE(B:INT  FS:FIELD_SPECS):
                BEGIN   VAR BIT=UCODE_BIT;
                BIT:=BIT(B);
                @(BIT).EQUATION:=FS;
                END
        ENDDEFN

        DEFINE(13,B2ADR\IS 7);
        DEFINE(14,ALU\IS 'X0X0');
        DEFINE(15,ALU\IS '0XXX'\AND BIT(14)\IS 0);
        DEFINE(17,ALU\IS 'X000');
        DEFINE(18,ALU\IS 'X111');
        DEFINE(19,ALU\IS '10XX');
        "BITS 16 AND 20 ARE DEFINED WITH THE ALU"

"Well, lets get on with it..."
```

```
IO_LEFT((READ_FROM_UPPER_WHEN(B2ADR\IS 7\AND B2RW\IS 1\BUFFERED_BY P1MUX);
        WRITE_TO_UPPER_WHEN(BIT(13)\IS 0\AND WRSRC\IS 0\BUFFERED_BY P1MUX);
        REFRESH_WHEN(PHI_2);
        WRITE_TO_LOWER_WHEN(B1ADR\IS 7\BUFFERED_BY P1MUX)),
        CONTROL\TO_PAD IN\BUFFERED_BY P1MUX,
        CONTROL\TO_PAD IN\BUFFERED_BY P1MUX,
        'PORT');

MEMORY(6,(READ_FROM_UPPER_WHEN(B2ADR\DECODE_STARTING_WITH 1\AND B2RW\IS 1\BUFFERE
        WRITE_TO_UPPER_WHEN(B2ADR\DECODE_STARTING_WITH 1\AND B2RW\IS 0\BUFFERED_B
        REFRESH_WHEN(PHI_2);
        WRITE_TO_LOWER_WHEN(B1ADR\DECODE_STARTING_WITH 1\BUFFERED_BY P1MUX)),
        'REGISTERS');

ALU(    (READ_FROM_UPPER_WHEN(PHI_1)),
        (READ_FROM_LOWER_WHEN(PHI_1)),
        (WRITE_TO_UPPER_WHEN(B2RW\IS 1\AND WRSRC\IS 0\BUFFERED_BY P1MUX)),
        ( "FUNCTIONS:"
        ALU\IS 'X0XX'\BUFFERED_BY P2MUX\NAMED 'PTT';
        ALU\IS 'XX0X'\BUFFERED_BY P2MUX\NAMED 'PTF';
        BIT(16)\IS 0\BUFFERED_BY P2MUX\NAMED 'PFT';
        BIT(15)\IS 0\AND ALU\IS 'XX1X'\WHICH_ALSO_DEFINES BIT(20)\BUFFERED_BY P2M
        ALU\IS '10XX'\BUFFERED_BY P2MUX\NAMED 'KTT';
        ALU\IS '1X0X'\BUFFERED_BY P2MUX\NAMED 'KTF';
        INTERNAL\IS 'XXXX000X'\WHICH_ALSO_DEFINES BIT(16)\BUFFERED_BY P2MUX\NAMED
        BIT(20)\IS 0\BUFFERED_BY P2MUX\NAMED 'KFF';
        PHI_1\NAMED 'PRE';
        "COUT" UNUSED;
        "CN-1" UNUSED;
        GND\NAMED 'RFF';
        VDD\NAMED 'RFT';
        VDD\NAMED 'RTF';
        GND\NAMED 'RTT';
        ALU\IS '00XX'\BUFFERED_BY P2MUX\NAMED 'CIN';
        PAD\TO_PAD OUT\BUFFERED_BY BUFOUT\NAMED 'MSB';
        PAD\TO_PAD OUT\BUFFERED_BY BUFOUT\NAMED 'ZERO';
        "WB"  UNUSED;
        PHI_2\NAMED 'WA'),
        'ALU');

PRECHARGE_BOTH(PHI_2);

SPACER;

OUT_RIGHT((READ_FROM_UPPER_WHEN(B2ADR\IS 0\AND B2RW\IS 1\BUFFERED_BY P1MUX);
        WRITE_TO_UPPER_WHEN(B2ADR\IS 0\AND B2RW\IS 0\BUFFERED_BY P1MUX);
        REFRESH_WHEN(PHI_2);
        WRITE_TO_LOWER_WHEN(B1ADR\IS 0\BUFFERED_BY P1MUX)),
        'I.R.');

CHIP('PASCAL');

BUILD_DATAPATH;

^DD_BUFFERS;

ADD_PLSR(PAD\TO_PAD IN\NAMED 'PLSR IN',PAD\TO_PAD OUT\NAMED 'PLSR OUT');

"WE DONT HAVE THE DECODER YET, BUT WE WILL SAY ADD_INSTRUCTION_DECODER;"
```

```
ASSEMBLE_CHIP;

RENAME('PASCAL:_CHIP'\FILE);


"PRINT SOME STATISTICS FOR DICK IN THE FILE PASCAL.ICG:"
        VAR C=CONNECT;SP=STRETCH_POINT;
        /*OUT_LOG PASCAL;*/
        WRITE(+1 FOR C $E CHIP.PAD_CONNECTS;);
        WRITE(' PAD CONNECTION POINTS:');
        CRLF;
        DO      WRITE(C.FROM.X\INT);
                TAB;
                WRITE(C.FROM.Y\INT);
                TAB;
                WRITE(C.NAME);
                WRITE(' #');
                WRITE(C.INDEX);
                CRLF;   FOR C $E CHIP.PAD_CONNECTS;
        CRLF;
        WRITE(+1 FOR C $E CHIP.DECODER_CONNECTS;);
        WRITE(' DECODER CONNECTION POINTS:');
        CRLF;
        DO      WRITE(C.FROM.X\INT);
                TAB;
                WRITE(C.FROM.Y\INT);
                TAB;
                WRITE(C.NAME);
                WRITE(' #');
                WRITE(C.INDEX);
                CRLF;   FOR C $E CHIP.DECODER_CONNECTS;
        CRLF;
        WRITE(+1 FOR SP $E STRETCH_POINTS;);
        WRITE(' STRETCH POINTS:');
        CRLF;
        DO      WRITE(SP\INT);
                TAB;
                TYPE(SP);
                CRLF;   FOR SP $E STRETCH_POINTS;
        /*CLOSE PASCAL.ICG;*/
```

A P E N D I X ------- D

MICRO CODE INPUT

```
field zspare = 1 ! to make the outputs even!!!!

field nasrc = 3,4
 constant jmpindr nasrc=1
 constant ret nasrc=2

field cc = 5,16
 constant cclsb cc=8
 constant ccmsb cc=2
 constant cczero cc=4
 constant ccrdy cc=1

field hl = 1,1
 constant call hl=0
 constant subr hl=0

field aluop = 6,0
 constant inc aluop=44
 constant dec aluop=3
 constant add aluop=6
 constant sub aluop=41
 constant and aluop=24
 constant or aluop=30
 constant not aluop=19
 constant pass aluop=26
 constant zero aluop=16
 constant ones aluop=31

field b1src = 3,0
 constant b1rda b1src=0
 constant b1rdb b1src=1
 constant b1rdbase b1src=2
 constant b1rdtop b1src=3
 constant b1rdpc b1src=4
 constant b1rdir b1src=5
 constant b1rdtmp b1src=6
 constant b2wrport b1src=7

field b2src = 4,0
 constant b2rda b2src=0
 constant b2rdb b2src=1
 constant b2rdbase b2src=2
 constant b2rdtop b2src=3
 constant b2rdpc b2src=4
 constant b2rdir b2src=5
 constant b2rdtmp b2src=6
 constant b2wra b2src=8
 constant b2wrb b2src=9
 constant b2wrbase b2src=10
 constant b2wrtop b2src=11
 constant b2wrpc b2src=12
 constant b2wrir b2src=13
 constant b2wrtmp b2src=14

field wrsrc = 1,0
 constant b2rdalu wrsrc=0
 constant b2rdport wrsrc=1

field buscntl = 3,0
```

```
 constant memrdcode buscntl=4
 constant memrddata buscntl=5
 constant memwrdata buscntl=6

output hl,nextaddress,nasrc,cc,b1src,b2src,wrsrc,aluop,buscntl
 utput zspare

code reserve 15 begin


0    :reset  :        : zero
     :       :        : b2rdalu,b2wrpc,zero                ! pc <- 0
     :       :        : b2rdalu,b2wrbase,zero              ! base <- 0
     :       :ifetch  : b2rdalu,b2wrtop                    ! top <- 0

1    :add    :svrslt  : b1rda,b2rdb,add

2    :sub    :svrslt  : b1rda,b2rdb,sub

3    :and    :svrslt  : b1rda,b2rdb,and

4    :or     :svrslt  : b1rda,b2rdb,or

5    :not    :        : b2rda,not
     :       :ifetch  : b2rdalu,b2wra

6    :lit    :        : b2rdpc,b2wrport,inc,memrdcode      ! port <- @pc
even:        :entrl   : b2rdalu,b2wrpc,call
     :       :ifetch  : b2rdport,b2wra                     ! a <- port

     :lod    :        : b2rdpc,b2wrport,inc,memrdcode      ! ir <- @pc
even:        :entrl   : b2rdalu,b2wrpc,call
     :       :        : b1rdbase,pass                      ! tmp <- base
even:        :chain   : b2rdalu,b2wrtmp,call               ! alu <- ea
     :       :        : b2rdalu,b2wrport,memrddata
     :       :        :                                    ! nop
even:h       :h       : ccrdy
     :       :ifetch  : b2rdport,b2wra                     ! a <- memleal

8    :sto    :        : b2rdpc,b2wrport,inc,memrdcode      ! ir <- @pc
even:        :entrl   : b2rdalu,b2wrpc,call
     :       :        : b1rdbase,pass                      ! tmp <- base
even:        :chain   : b2rdalu,b2wrtmp,call               ! alu <- ea
     :       :        : b2rdalu,b2wrport,memwrdata         ! memleal <- a
     :       :        : b2rda,b2wrport
even:l       :l       : ccrdy
     :       :ifetch  :

9    :call   :        : b2rdpc,b2wrport,inc,memrdcode      ! ir <- @pc
even:        :entrl   : b2rdalu,b2wrpc,call
     :       :        : b1rdbase,pass
     :       :        : b2rdalu,b2wrb,b1rdpc,pass          ! b <- base
     :       :        : b2rdalu,b2wra,b1rdtop,pass         ! a <- pc
     :       :        : b2rdalu,b2wrbase,b1rdir,pass       ! base <- top
     :       :ifetch  : b2rdalu,b2wrpc                     ! pc <- ir

 J   :ret    :        : b2rdbase,dec
     :       :        : b2rdalu,b2wrtop,b1rdb,pass         ! top <- base-1
     :       :        : b2rdalu,b2wrbase,b1rda,pass        ! base <- b
     :       :ifetch  : b2rdalu,b2wrpc                     ! pc <- a
```

```
11   :eq      :        : b1rda,b2rdb,sub
     :        :        : zero,cczero
even:svrslt :ifetch   : b2rdalu,b2wrb                          ! b <- 0
     :zro     :svrslt  : ones                                  ! b <- 1

12   :jct     :        : b2rdpc,b2wrport,inc,memrdcode          ! ir <- @pc
even:         :entrl   : b2rdalu,b2wrpc,call
     :        :        : b1rdir,pass,cc1sb                     ! lsb set by 'entrl'
even:nj       :ifetch  :
     :j       :ifetch  : b2rdalu,b2wrpc                        ! pc <- ir

13   :push    :        : b2rdtop,inc                           ! top <- top+1
     :        :        : b2rdalu,b2wrtop,b2wrport,memwrdata
     :        :        : b2rdb,b2wrport                        ! mem[top] <- b
even:x        :x       : ccrdy
     :        :svrslt  : b1rda,pass                            ! b <- a

14   :pop     :        : b1rdb,pass
     :        :        : b2rdalu,b2wra                         ! a <- b
     :        :        : b2rdtop,b2wrport,dec,memrddata        ! top <- top-1
even:         :entrl   : b2rdalu,b2wrtop,call                  ! b <- mem[top]
     :        :ifetch  : b2rdport,b2wrb


! This is the entry point to start instruction execution.
!
even:ifetch  :fatpc   : b2rdpc,b2wrport,inc,memrdcode,call!  ir <- @pc
     :        :        : jmpindr


! This subroutine fetches the byte pointed to by the PC into the IR and then
! increments the PC.
!
     :fatpc   :        : b2rdalu,b2wrpc,subr                   ! pc <- pc+1
even:entrl    :entrl   : ccrdy,subr                            ! wait for RDY
     :        :        : b2rdport,b2wrir,b1rda,pass,ret,subr

! This subroutine follows the pointer chain pointed to by TMP for the number
! of levels in IR, then fetches the OFFSET incrementing PC, and returns the
! effective address in ALU.
!
     :chain   :        : b2rdir,dec,sub                        ! ir <- ir-1
     :        :ndone   : b2rdalu,b2wrir,ccmsb,subr
     :back    :        : subr
even:wt       :wt      : ccrdy,subr
     :        :chain   : b2rdport,b2wrtmp,subr
even:ndone    :back    : b2rdtmp,b2wrport,memrddata,subr
     :done    :        : b2rdpc,b2wrport,inc,memrdcode,subr! port <- <OFF>
     :        :        : b2rdalu,b2wrpc,subr
even:w        :w       : ccrdy,subr
     :        :        : b1rdtmp,b2rdport,add,ret,b2wrir,subr! alu <- OFF+tmp

end
```

--- F I E L D S -----

ALUOP	6	0
B1SRC	3	0
B2SRC	4	0
BUSCNTL	3	0
CC	5	16
HL	1	1
NASRC	3	4
WRSRC	1	0
ZSPARE	1	0

--- C O N S T A N T S -----

ALUOP	6	0
ADD	6	
AND	24	
DEC	3	
INC	44	
NOT	19	
ONES	31	
OR	30	
PASS	26	
SUB	41	
ZERO	16	
B1SRC	3	0
B1RDA	0	
B1RDB	1	
B1RDBASE	2	
B1RDIR	5	
B1RDPC	4	
B1RDTMP	6	
B1RDTOP	3	
B2WRPORT	7	
B2SRC	4	0
B2RDA	0	
B2RDB	1	
B2RDBASE	2	
B2RDIR	5	
B2RDPC	4	
B2RDTMP	6	
B2RDTOP	3	
B2WRA	8	
B2WRB	9	
B2WRBASE	10	
B2WRIR	13	
B2WRPC	12	
B2WRTMP	14	
B2WRTOP	11	
BUSCNTL	3	0
MEMRDCODE	4	
MEMRDDATA	5	
MEMWRDATA	6	
CC	5	16
CCLSB	8	

```
          CCMSB                    2
          CCRDY                    1
          CCZERO                   4

      HL                      1          1
          CALL                     0
          SUBR                     0

      NASRC                   3          4
          JMPINDR                  1
          RET                      2

      WRSRC                   1          0
          B2RDALU                  0
          B2RDPORT                 1


     --- O U T P U T F O R M A T
MINTERMS= 74 INPUTS=  7 OUTPUTS= 34
input: ADDR
      HL
      NEXTADDRESS
      NASRC
      CC
      B1SRC
      B2SRC
      WRSRC
      ALUOP
      BUSCNTL
      ZSPARE


     --- C  O  D  E   ------ base =  16

 adr   nad   Label   Glabel    ALUOP   B1SRC   B2SRC   BUSCNTL   CC   HL   NASRC
   0:   16: RESET              16       0       0       0      16    1      4
   1:   48: ADD      SVRSLT     6       0       1       0      16    1      4
   2:   40: SUB      SVRSLT    41       0       1       0      16    1      4
   3:   48: AND      SVRSLT    24       0       1       0      16    1      4
   4:   48: OR       SVRSLT    30       0       1       0      16    1      4
   5:   19: NOT                19       0       0       0      16    1      4
   6:   20: LIT                44       7       4       4      16    1      4
   7:   22: LOD                44       7       4       4      16    1      4
   8:   30: STO                44       7       4       4      16    1      4
   9:   38: CALL               44       7       4       4      16    1      4
  10:   44: RET                 3       0       2       0      16    1      4
  11:   47: EQ                 41       0       1       0      16    1      4
  12:   50: JCT                44       7       4       4      16    1      4
  13:   54: PUSH               44       0       3       0      16    1      4
  14:   58: POP                26       1       0       0      16    1      4
  16:   17:                    16       0      12       0      16    1      4
  17:   18:                    16       0      10       0      16    1      4
  18:   62:           IFETCH    0       0      11       0      16    1      4
  19:   62:           IFETCH    0       0       8       0      16    1      4
  20:   66:           ENTRL     0       0      12       0      16    0      4
  21:   62:           IFETCH    0       0       8       0      16    1      4
  22:   66:           ENTRL     0       0      12       0      16    0      4
  23:   24:                    26       2       0       0      16    1      4
  24:   68:           CHAIN     0       0      14       0      16    0      4
  25:   26:                     0       7       0       5      16    1      4
```

26:	28:			0	0	0	0	16	1	4
28:	28:	H	H	0	0	0	0	1	1	4
29:	62:		IFETCH	0	0	8	0	16	1	4
30:	66:		ENTRL	0	0	12	0	16	0	4
31:	32:			26	2	0	0	16	1	4
32:	68:		CHAIN	0	0	14	0	16	0	4
33:	34:			0	7	0	6	16	1	4
34:	36:			0	7	0	0	16	1	4
36:	36:	L	L	0	0	0	0	1	1	4
37:	62:		IFETCH	0	0	0	0	16	1	4
38:	66:		ENTRL	0	0	12	0	16	0	4
39:	40:			26	2	0	0	16	1	4
40:	41:			26	4	9	0	16	1	4
41:	42:			26	3	8	0	16	1	4
42:	43:			26	5	10	0	16	1	4
43:	62:		IFETCH	0	0	12	0	16	1	4
44:	45:			26	1	11	0	16	1	4
45:	46:			26	0	10	0	16	1	4
46:	62:		IFETCH	0	0	12	0	16	1	4
47:	48:			16	0	0	0	4	1	4
48:	62:	SVRSLT	IFETCH	0	0	9	0	16	1	4
49:	48:	ZRO	SVRSLT	31	0	0	0	16	1	4
50:	66:		ENTRL	0	0	12	0	16	0	4
51:	52:			26	5	0	0	8	1	4
52:	62:	NJ	IFETCH	0	0	0	0	16	1	4
53:	62:	J	IFETCH	0	0	12	0	16	1	4
54:	55:			0	7	11	6	16	1	4
55:	56:			0	7	1	0	16	1	4
56:	56:	X	X	0	0	0	0	1	1	4
57:	48:		SVRSLT	26	0	0	0	16	1	4
58:	59:			0	0	8	0	16	1	4
59:	60:			3	7	3	5	16	1	4
60:	66:		ENTRL	0	0	11	0	16	0	4
61:	62:		IFETCH	0	0	9	0	16	1	4
62:	64:	IFETCH	FATPC	44	7	4	4	16	0	4
63:	64:			0	0	0	0	16	1	1
64:	66:	FATPC		0	0	12	0	16	0	4
66:	66:	ENTRL	ENTRL	0	0	0	0	1	0	4
67:	68:			26	0	13	0	16	0	2
68:	69:	CHAIN		41	0	5	0	16	1	4
69:	74:		NDONE	0	0	13	0	2	0	4
70:	72:	BACK		0	0	0	0	16	0	4
72:	72:	WT	WT	0	0	0	0	1	0	4
73:	68:		CHAIN	0	0	14	0	16	0	4
74:	70:	NDONE	BACK	0	7	6	5	16	0	4
75:	76:	DONE		44	7	4	4	16	0	4
76:	78:			0	0	12	0	16	0	4
78:	78:	W	W	0	0	0	0	1	0	4
79:	80:			6	6	13	0	16	0	2

```
00000000 10010000100100000000000000100000000
00000001 10110000100100000000000100001100000
00000010 10110000100100000000001010100100000
00000011 10110000100100000000001001100000000
00000100 10110000100100000000001001111000000
00000101 10010011100100000000000001001100000
00000110 10010100100100001111010001011001000
00000111 10010110100100001111010001011001000
00001000 10011110100100001111010001011001000
00001001 10100110100100001111010001011001000
00001010 10101100100100000000001000000110000
00001011 10101111100100000000001010100100000
00001100 10110010100100001111010001011001000
00001101 10110110100100000000001101011000000
00001110 10111010100100000010000001101000000
00010000 10010001100100000001100001000000000
00010001 10010010100100000001010001000000000
00010010 10111111010010000001011000000000000
00010011 10111111010010000000100000000000000
00010100 01000010100100000001100000000000000
00010101 10111111010010000000100010000000000
00010110 01000010100100000001100000000000000
00010111 10011000100100000100000001101000000
00011000 01000100100100000001110000000000000
00011001 10011010100100001110000000000001010
00011010 10011100100100000000000000000000000
00011100 10011100100000001000000000000000000
00011101 10111110100100000001000100000000000
00011110 01000010100100000001100000000000000
00011111 10100000100100000100000001101000000
00100000 01000100100100000001110000000000000
00100001 10100010100100001110000000000001100
01000010 10100100100100001110000000000000000
01001000 10100100100000001000000000000000000
01001001 10111111010010000000000000000000000
01001010 01000010100100000001100000000000000
01001011 10101000100100000100000001101000000
01001100 10100100100100001110000000000000000
01001101 10101000100100000100000001101000000
01010000 10100100110010000100100100110100000
01010001 10101010100100000111000001101000000
01010010 10101011100100001011010001101000000
01010011 10111111010010000001100000000000000
01011000 10101101100100000011011001101000000
01011001 10101110100100000001010001101000000
01011010 10111111010010000001100000000000000
01011011 10110000100001000000000001000000000
01100000 10111111010010000000100100000000000
01100001 10110000100100000000000000111110000
01100010 01000010100100000001100000000000000
01100011 10110100100010001010000001101000000
01100100 10111111010010000000000000000000000
01100101 10111111010010000000100000000000000
01100110 10111011100100001111011000000001100
01100111 10111000100100001110001000000000000
01101000 10111000100000001000000000000000000
01101001 10110000100100000000000001101000000
01101010 10111011100100000100000000000000000
01101011 10111111010010000000100000000000000
01101100 10111111010010000000100000000000000
01101101 10111000100100000100000001101000000
01101110 10111111010010000000100000000000000
01101111 10110000100100000000000000111110000
01110000 10111111010010000000100000000000000
01110001 10110000100100000000000000111110000
01110010 01000010100100000001100000000000000
01110011 10110100100010001010000001101000000
01110100 10111111010010000000000000000000000
01110101 10111111010010000000100000000000000
01110110 10111011100100001111011000000001100
01110111 10111000100100001110001000000000000
01111000 10111000100000001000000000000000000
01111001 10110000100100000000000001101000000
01111010 10111011100100000100000000000000000
01111011 10111110010010000111001100000111010
01111100 01000010100100000001011000000000000
01111101 10111111010010000000100110000000000
01111110 01000000100100001110100010110001000
```

```
0111111 11000000001100000000000000000000000
1000000 01000010100100000001100000000000000
1000010 01000010100000001000000000000000000
1000011 01000100010100000001101101101000000
1000100 11000101100100000000101010100100000
 000101 01001010100000100001101000000000000
1000110 01001000100100000000000000000000000
1001000 01001000100000001000000000000000000
1001001 01000100100100000001110100000000000
1001010 01000110100100001110110000000001010
1001011 01001100100100001110100010110010000
1001100 01001110100100000001100000000000000
1001110 01001110100000001000000000000000000
1001111 01010000010100001101101100011000000
```

```
BEGIN
    EXTERNAL CLASS jeddef,fork,io20,tty,td850,commandline;
    EXTERNAL INTEGER PROCEDURE imin,imax,checkint;
    EXTERNAL TEXT PROCEDURE maketext,conc,scanto;
    EXTERNAL TEXT PROCEDURE front,rest;
    EXTERNAL PROCEDURE enterdebug,depchar;
    EXTERNAL BOOLEAN PROCEDURE jsys,skipin,menu,lookup,frontcompare;
    EXTERNAL INTEGER PROCEDURE xwd,right,left,land,lor,lnot,taddr,aaddr,lshift;
    EXTERNAL TEXT PROCEDURE getitem,conc2,frontstrip,upcase;
    EXTERNAL TEXT PROCEDURE today,daytime;
    EXTERNAL CHARACTER PROCEDURE fetchar;
    simset
    BEGIN
        TEXT buf;
        TEXT token;
        REF(io20) inf;
        REF(io20) pntf;
        REF(io20) outf;
        REF(commandline)cmdline;
        TEXT ARRAY names[1:200];
        REF(stuff) ARRAY stuffs[1:200];
        REF(stuff) ARRAY outputs[1:200];
        REF(field) ARRAY fields[1:200];
        INTEGER lastoutputs;
        INTEGER laststuff;
        INTEGER lastfield;
        INTEGER codebase;

        CLASS stuff(nam); TEXT nam;
        VIRTUAL: PROCEDURE showmyself;
        PROCEDURE print;
        PROCEDURE output;
        INTEGER PROCEDURE lgh;
        INTEGER PROCEDURE fieldnum;
        BOOLEAN PROCEDURE  used;
        BOOLEAN PROCEDURE setused;
        BEGIN
            BOOLEAN meused;
            BOOLEAN PROCEDURE setused;
            BEGIN
                setused:=meused;
                meused:=TRUE;
            END of setused;
            BOOLEAN PROCEDURE used;
            used:=meused;
            INTEGER PROCEDURE num;
            BEGIN
                INTEGER i,j,k;
                IF lookup(nam,names,1,laststuff,i) THEN ELSE j:=j/0;
                num:=i;
            END of num;

            BOOLEAN PROCEDURE enter;
            BEGIN INTEGER i,j,k;
                nam:-copy(nam);nam:-upcase(nam);
                IF laststuff=0 THEN
                BEGIN
                    laststuff:=laststuff+1;
                    names[laststuff]:-nam;
                    stuffs[laststuff]:-THIS stuff;
```

```
        END ELSE
        BEGIN
            IF lookup(nam,names,i,laststuff,i) THEN enter:=TRUE ELSE
            BEGIN
                i:=i;
                WHILE nam > names[i] AND i <= laststuff DO i:=i+1;
                IF i > laststuff THEN
                BEGIN
                    laststuff:=laststuff+1;
                    names[laststuff]:-nam;
                    stuffs[laststuff]:-THIS stuff;
                END ELSE
                BEGIN
                    j:=laststuff;
                    WHILE j >= i DO
                    BEGIN
                        names[j+1]:-names[j];
                        stuffs[j+1]:-stuffs[j];
                        j:=j-1;
                    END;
                    names[i]:-nam;
                    stuffs[i]:-THIS stuff;
                    laststuff:=laststuff+1;
                END;
            END;
        END;
    END of enter;
END of stuff;

stuff CLASS nextaddress;
BEGIN
    INTEGER PROCEDURE lgh;
    BEGIN
        lgh:=7;
    END;
    enter;
END of nextaddress;

stuff CLASS field;
BEGIN
    INTEGER llgh;
    INTEGER defaultval;
    INTEGER PROCEDURE lgh;
    BEGIN
        lgh:=llgh;
    END of lgh;
    INTEGER PROCEDURE size;
    BEGIN REAL turkey;! this is for this screw ball machine;
        turkey:=lgh;
        turkey:=2**turkey-1;
        size:=turkey;
    END;
    INTEGER PROCEDURE fieldnum;
    BEGIN
        INTEGER i;
        i:=1; WHILE fields[i] =/= THIS field DO i:=i+1;
        fieldnum:=i;
    END;
    PROCEDURE showmyself;
    BEGIN
```

```
        outtext(nam);outtext(" = ");outint(lgh,3);
        outtext(",");outint(defaultval,3);
        outimage;
    END of showmyself;
    PROCEDURE print;
    BEGIN INTEGER i;
        INSPECT pntf DO
        BEGIN
            outtext("     ");
            outtext(nam);
            i:=20-nam.length;
            WHILE i > 1 DO BEGIN i:=i-1;outchar(' ');END;
            outint(lgh,3);
            outtext("     ");outint(defaultval,3);
            outimage;
        END;
    END of outpnt;
END of class field;

stuff CLASS constant;
BEGIN
    INTEGER val;
    REF(field) fld;
    PROCEDURE showmyself;
    BEGIN
        outtext(nam);outtext(" ");IF fld =/= NONE THEN
        outtext(fld.nam);outtext(" ");
        outtext(" = ");outint(val,3);
        outimage;
    END of showmyself;
    PROCEDURE print;
    BEGIN INTEGER i;
        INSPECT pntf DO
        BEGIN
            outtext("          ");
            outtext(nam);
            i:=20-nam.length;
            WHILE i > 1 DO BEGIN i:=i-1;outchar(' ');END;
            outint(val,3);
            outimage;
        END;
    END of outpnt;
END of class constant;

head CLASS bigmac;
BEGIN
END of bigmac;

REF(bigmac) thecode;

TEXT PROCEDURE transbin(x,L);VALUE x,L;INTEGER x,L;
BEGIN
    INTEGER i,j;
    TEXT t;
    t:-blanks(L);
    FOR i:=1 STEP 1 UNTIL L DO
    BEGIN
        IF land(x,1) > 0 THEN
        depchar(t,L-i+1,'1') ELSE depchar(t,L-i+1,'0');
        x:=x//2;
```

```
    END;
    transbin:-t;
END of transbin;

link CLASS microinst;
BEGIN
    INTEGER adr;
    INTEGER nextadr;
    TEXT lab,golab;
    ARRAY fieldvals[1:lastfield];
    PROCEDURE print;
    BEGIN
        INTEGER i,j,k;
        INSPECT pntf DO
        BEGIN
            outint(adr,4);outchar(':');outint(nextadr,4);outchar(':');
            IF lab NE NOTEXT THEN
            BEGIN
                IF lab.length > 7 THEN lab:-lab.sub(1,7);
                outchar(' ');
                outtext(lab);i:=7-lab.length;
                WHILE i> 1 DO BEGIN outchar(' ');i:=i-1;END;
                outchar(' ');
            END ELSE
            FOR i:=1 STEP 1 UNTIL 8 DO
            outchar(' ');
            IF golab NE NOTEXT THEN
            BEGIN
                IF golab.length > 7 THEN golab:-golab.sub(1,7);
                outchar(' ');
                outtext(golab);i:=7-golab.length;
                WHILE i> 1 DO BEGIN outchar(' ');i:=i-1;END;
                outchar(' ');
            END ELSE
            FOR i:=1 STEP 1 UNTIL 8 DO
            outchar(' ');
            FOR i:=1 STEP 1 UNTIL lastfield DO
            BEGIN
                outchar( ' ');
                outint(fieldvals[i],3);
                j:=fields[i].nam.length;j:=imax(3,j);
                j:=imin(8,j);
                WHILE j> 1 DO BEGIN outchar(' ');j:=j-1;END;
            END;
            outimage;
        END;
    END of print;
    PROCEDURE output;
    BEGIN
        INTEGER i,j;
        INSPECT outf DO
        BEGIN
            outtext(transbin(adr,7));
            outtext(" ");
            FOR i:=1 STEP 1 UNTIL lastoutputs DO
            IF outputs[i] IS nextaddress THEN
            BEGIN
                outtext(transbin(nextadr,7));
            END ELSE
            outtext(transbin(fieldvals[outputs[i].fieldnum],
```

```
            outputs[i].lgh));
            outimage;
        END;
    END of output;
    PROCEDURE init;
    BEGIN
        INTEGER i;
        FOR i:=1 STEP 1 UNTIL lastfield DO
        fieldvals[i]:=fields[i].defaultval;
    END of init;
    init;
END of microinst;

BOOLEAN PROCEDURE nextline;
BEGIN
    IF NOT inf.endfile THEN
    BEGIN
        buf:=inf.image.strip;buf.setpos(1);
        inf.inimage;
    END ELSE nextline:=TRUE;
END of nextline;

TEXT PROCEDURE nexttoken;
BEGIN
    token:-getitem(buf);upcase(token);
    nexttoken:-token;
END of nexttoken;

BOOLEAN PROCEDURE getnum(n);NAME n;INTEGER n;
BEGIN
    INTEGER i;
    IF checkint(token) > 0 THEN
    BEGIN
        n:=token.getint;nexttoken;
    END ELSE
    getnum:=TRUE;
END of getnum;

PROCEDURE error(t);VALUE t;TEXT t;
BEGIN INTEGER i;
    outtext(buf);outimage;
    i:=buf.pos;WHILE i > 1 AND i<70 DO BEGIN i:=i-1;outchar(' ');END;
    outchar('^');
    outimage;outtext(t);outimage;
END of error;

PROCEDURE processdec;
BEGIN
    INTEGER i;
    REF(field) f;
    REF(constant) c;
    WHILE NOT nextline AND nexttoken NE "CODE" DO
    BEGIN
        IF token = "FIELD" THEN
        BEGIN
            f:-NEW field(nexttoken);
            IF nexttoken NE "=" THEN
            BEGIN error(" = expected");
                GO TO nextstuff;
            END;
```

```
        nexttoken;
        IF getnum(i) THEN BEGIN error("number expected"); GO TO
        nextstuff;END;
        f.llgh:=i;
        IF token="," THEN
        BEGIN
            nexttoken;
            IF getnum(i) THEN BEGIN error("number expected");GO TO
            nextstuff;END;
            f.defaultval:=i;
        END;
        IF f.enter THEN error("already declared");
    END ELSE
    IF token="CONSTANT" THEN
    BEGIN
        c:-NEW constant(nexttoken);
        nexttoken;
        IF lookup(token,names,i,laststuff,i) THEN
        BEGIN
            IF stuffs[i] IS field THEN
            c.fld:-stuffs[i] QUA field ELSE
            BEGIN
                error(" not a field");
                GO TO nextstuff;
            END;
        END
        ELSE
        BEGIN
            error(" undefined field");
            GO TO nextstuff;
        END;
        IF nexttoken NE "=" THEN error(
        "equal expected") ELSE
        nexttoken;
        IF getnum(i) THEN BEGIN error("a number was expected");GO
        TO nextstuff;END;
        IF c.fld =/= NONE THEN
        BEGIN
            IF i > c.fld.size THEN
            BEGIN
                i:=c.fld.size;
                error(
                " the constant is greater than the field size");
            END;
        END;
        c.val:=i;
        IF c.enter THEN error("already declared");
    END ELSE
    IF token="OUTPUT" THEN
    BEGIN
        nexttoken;
        morejunk: WHILE token="," DO nexttoken;
        IF lookup(token,names,i,laststuff,i) THEN
        BEGIN
            IF stuffs[i] IS field OR stuffs[i]IS nextaddress THEN
            BEGIN
                IF stuffs[i].setused THEN
                error("this field was used before!!");
                lastoutputs:=lastoutputs+1;
                outputs[lastoutputs]:-stuffs[i];
```

```
                END
                ELSE
                error("NOT A FIELD or nextaddress");
            END
            ELSE
            error("NOT A FIELD or nextaddress");
            IF nexttoken NE NOTEXT THEN GO TO morejunk;
        END ELSE
        IF token = "!"  THEN ELSE
        IF token= NOTEXT THEN ELSE ! special for big GGGGGGGGGGGGG;
        IF token NE "CODE" THEN error(" undefined stuff");
        nextstuff:
    END;
    FOR i:=1 STEP 1 UNTIL laststuff DO
    BEGIN
        IF stuffs[i] IS field THEN
        BEGIN
            lastfield:=lastfield+1;
            fields[lastfield]:-stuffs[i] QUA field;
        END;
    END;
    exit:
END of processdec;

PROCEDURE processcode;
BEGIN
    INTEGER ci,i,j,k,last;
    BOOLEAN b;
    REF(microinst) inst,si,sl;
    thecode:-NEW bigmac;
    IF token NE "CODE" THEN ERROR(" the keyword code expected") ELSE
    nexttoken;
    IF token NE "RESERVE" THEN error(" the keyword reserve expected")
    ELSE  nexttoken;
    IF getnum(i) THEN error(" a number was expected") ELSE
    codebase:=last:=i+1;
    IF token NE "BEGIN" THEN
    error(" the keyword begin expected") ELSE
    nexttoken;
    WHILE NOT nextline AND nexttoken NE "END" DO
    BEGIN
        IF token=NOTEXT or token="!" THEN
        WHILE NOT nextline AND (nexttoken = NOTEXT OR token="!")DO;
        if token="END" then go to exitthisstuff;
        sl:-inst;
        inst:-NEW microinst;

        ! determine current address for this op;
        IF checkint(token) > 0 THEN
        BEGIN ! user defined;
            ci:=token.getint;nexttoken;
            b:=TRUE;
        END ELSE
        IF token="EVEN" THEN
        BEGIN ! userforced to even;
            IF mod(last,2)=1 THEN
            BEGIN
                last:=last+1;! set even;
                sl.nextadr:=last;
            END;
```

```
    ci:=last;
    last:=last+i;
    nexttoken;
END ELSE
BEGIN
    ! before we use last lets try to look for a hole;
    k:=i:=0;
    si:-thecode.first;
    WHILE si =/= NONE DO
    BEGIN
        j:=i;i:=si.adr;si:-si.suc;
        IF j > codebase THEN
        BEGIN
            IF i-j>2 THEN ! must have a hole ;
            BEGIN
                si:-NONE;
                k:=j+i;
            END;
        END;
    END;
    IF k NE 0 THEN ci:=k ELSE
    BEGIN
        ci:=last;! we may need some adjustment   here;
        last:=last+i;
    END;
END;
inst.adr:=ci;
IF token NE ":" THEN error(" a colon was expected") ELSE
nexttoken;

IF letter(fetchar(token,1)) THEN
BEGIN ! must be a label;
    si:-thecode.first;
    WHILE IF si =/= NONE THEN token NE si.lab ELSE FALSE DO
    si:-si.suc;
    IF si =/= NONE  THEN
    BEGIN
        error(" duplicate label !");
    END
    ELSE
    inst.lab:-copy(token);
    nexttoken;
END;
IF token NE ":" THEN error(" a colon was expected") ELSE
nexttoken;

IF checkint(token) > 0 THEN
BEGIN ! must be a next address;
    j:=token.getint;nexttoken;
    inst.nextadr:=j;
END ELSE
IF letter(fetchar(token,1)) THEN
BEGIN ! must be a label;
    inst.golab:-copy(token);
    nexttoken;
END ELSE
BEGIN
    inst.nextadr:=j:=last;
END;
IF token NE ":" THEN error(" a colon was expected") ELSE
```

```
    nexttoken;

    WHILE token NE NOTEXT AND token NE "!" DO
    BEGIN
        IF menu(token,j,names,laststuff) THEN
        BEGIN
            IF stuffs[j] IS constant THEN
            BEGIN
                inst.fieldvals[stuffs[j] QUA
                constant.fld.fieldnum]:=stuffs[j] QUA
                constant.val;
            END ELSE error(" not imp");
            nexttoken;
        END ELSE
        IF token="," THEN nexttoken ELSE
        BEGIN
            error(" a constant exp");
            nexttoken;
        END;
    END;


    ! now the code to add this inst;
    IF thecode.empty THEN inst.into(thecode) ELSE
    si:-thecode.first;
    WHILE si =/= NONE DO
    BEGIN
        IF si.adr=ci THEN
        BEGIN
            error(" same code address ");
            si:-NONE;
        END
        ELSE
        IF si.adr > ci THEN
        BEGIN
            inst.precede(si);si:-NONE;
        END ELSE
        IF si.suc =/= NONE THEN si:-si.suc ELSE
        BEGIN
            inst.follow(si);si:-NONE;
        END;
    END;
END;

exitthisstuff:

si:- thecode.first;
WHILE si =/= NONE DO
BEGIN
    IF si.golab =/= NOTEXT THEN
    BEGIN
        sl:-thecode.first;
        WHILE IF sl=/= NONE THEN sl.lab NE si.golab ELSE FALSE
        DO
        sl:-sl.suc;
        IF sl== NONE THEN
        BEGIN
            outtext(" label not found : ");outtext(si.golab);
            outimage;
        END ELSE
```

```
            si.nextadr:=sl.adr;
        END;
        si:-si.suc;
    END;
END of codeprocess;

PROCEDURE showmyself;
BEGIN
    INTEGER i;
    outtext("showmyself laststuff=");outint(laststuff,3);outimage;
    FOR i:=1 STEP 1 UNTIL laststuff DO
    stuffs[i].showmyself;
END of showmyself;

PROCEDURE print;
BEGIN
    INTEGER i,j,k;
    REF(field) f;
    REF(microinst) si;
    BOOLEAN b;
    INSPECT pntf DO
    BEGIN
        outtext("  --- F  I  E  L  D  S  -----");outimage;
        outtext(" ");outimage;
        FOR i:=1 STEP 1 UNTIL laststuff DO
        IF stuffs[i] IS field THEN stuffs[i].print;
        outtext(" ");outimage;
        outtext(" ");outimage;
        outtext("  --- C  O  N  S  T  A  N  T  S -----");outimage;
        FOR i:=1 STEP 1 UNTIL laststuff DO
        IF stuffs[i] IS field THEN
        BEGIN
            f:-stuffs[i] QUA field;
            b:=FALSE;
            FOR j:=1 STEP 1 UNTIL laststuff DO
            IF stuffs[j] IS constant THEN
            BEGIN
                IF stuffs[j] QUA constant.fld==f THEN
                BEGIN
                    IF NOT b THEN
                    BEGIN
                        outtext(" ");outimage;
                        f.print;
                        b:=TRUE;
                    END;
                    stuffs[j].print;
                END;
            END;
        END;
        outtext(" ");outimage;
        outtext(" ");outimage;
        outtext("    --- O U T P U T F O R M A T ");
        outtext(" ");outimage;
        si:-thecode.first;
        i:=0;
        WHILE si =/= NONE DO
        BEGIN
            i:=i+1;
            si:-si.suc;
        END;
```

```
            outtext("MINTERMS=");outint(i,3);
            outtext(" INPUTS=");outint(7,3);
            outtext(" OUTPUTS=");j:=0;
            FOR i:=1 STEP 1 UNTIL lastoutputs DO
            j:=j+outputs[i].lgh;
            outint(j,3);
            outtext(" ");outimage;
            outtext("input: ADDR");outimage;
            FOR i := 1 STEP 1 UNTIL lastoutputs DO
            BEGIN
                outtext("     ");
                outtext(outputs[i].nam);
                outimage;
            END;
            outtext(" ");outimage;
            outtext(" ");outimage;
            outtext("    --- C  O  D  E   ----- base = ");
            outint(codebase,3);
            outimage;
            outtext(" ");outimage;
            outtext(" adr ");outtext(" nad ");
            outtext("  Label ");
            outtext(" Glabel ");
            FOR i:=1 STEP 1 UNTIL lastfield DO
            BEGIN
                outchar(' ');outchar(' ');
                j:=fields[i].nam.length;
                IF j > 8 THEN outtext(fields[i].nam.sub(1,8)) ELSE
                outtext(fields[i].nam);
                WHILE j< 3 DO BEGIN j:=j+1;outchar(' ');END;
                outchar(' ');
            END;
            outimage;
            si:-thecode.first;
            WHILE si =/= NONE DO
            BEGIN
                si.print;
                si:-si.suc;
            END;
        END;
    END of print;

    PROCEDURE output;
    BEGIN
        REF(microinst) si;
        si:-thecode.first;
        WHILE si =/= NONE DO
        BEGIN
            si.output;
            si:-si.suc;
        END;
    END of output;

    ! This is the beginnning of the routine;
    sysout.image:-blanks(80);cmdline:-NEW commandline;
    buf:-blanks(80);
    outtext("Hi There Today is:");outtext(today);outtext(",Time:");
    outtext(daytime);
    outimage;
```

```
        NEW nextaddress(copy("NEXTADDRESS"));

        IF cmdline.nargs > 1 THEN
        BEGIN  ! a real cheap way to tell if we have a command line;
            TEXT filename,fs,fo,fp;
            filename:-copy(cmdline.args[2]).strip;
            filename:-conc2(filename,".");
            fs:-conc2(filename,"sym");
            fp:-conc2(filename,"pnt");
            fo:-conc2(filename,"cod");
            inf:-NEW io20(fs);
            IF inf.newfile THEN
            BEGIN
                inf.release;
                outtext(" No file name ");
                outimage;
                GO TO bigexit;

            END
            ELSE
            BEGIN
                inf.open(blanks(80));
                inf.inimage;
                pntf:-NEW io20(fp);
                pntf.write.open(blanks(120));
                outf:-NEW io20(fo);
                outf.write.open(blanks(80));
            END;
        END ELSE
        BEGIN
            outtext(" No file name was presented");
            outimage;
            GO TO bigexit;
        END;
        processdec;
        !showmyself;
        processcode;
        print;
        output;
        pntf.close;pntf.release;
        outf.close;outf.release;
        bigexit:
        outtext("good bye");outimage;
    END;
END;
```

A P E N D I X ------- E

DATA PATH DECODED INPUT

```
xxxxxxxxxxx101          1 B1RDIR
xxxxxxx1010xxx          2 B2RDIR
xxxxxxx1011xxx          3 B2WRIR
xxxxxx0xxx1xxx          4 B2RDALU
xxxx1xxxxxxxx           5 CIN
xxxxxxxxxxxxx           6 KFF
x0xxxxxxxxxxx           7 KFT
xx0x1xxxxxxxx           8 KTF
xxx01xxxxxxxx           9 KTT
1xxxxxxxxxxxx          10 PFF
x1xxxxxxxxxxx          11 PFT
xx1xxxxxxxxxx          12 PTF
xxx1xxxxxxxxx          13 PTT
zzzzzzzxxxx000         14 B1RDA
zzzzzzz0000xxx         15 B2RDA
zzzzzzz0001xxx         16 B2WRA
zzzzzzzxxxx100         17 B1RDB
zzzzzzz1000xxx         18 B2RDB
zzzzzzz1001xxx         19 B2WRB
zzzzzzzxxxx010         20 B1RDBASE
zzzzzzz0100xxx         21 B2RDBASE
zzzzzzz0101xxx         22 B2WRBASE
zzzzzzzxxxx110         23 B1RDTOP
zzzzzzz1100xxx         24 B2RDTOP
zzzzzzz1101xxx         25 B2WRTOP
zzzzzzzxxxx001         26 B1RDPC
zzzzzzz0010xxx         27 B2RDPC
zzzzzzz0011xxx         28 B2WRPC
zzzzzzzxxxx011         29 B1RDTMP
zzzzzzz0110xxx         30 B2RDTMP
zzzzzz0111xxx          31 B2WRTMP
xxxxxxxxxx111          32 B2WRPORT
xxxxxx1xxx1xxx         33 B2RDPORT
100100xxxxxxxx         34 B1RDPORT
```

INPUT ORDER:

```
   ALUOP      WRSRC  B2SRC  B2RW  B1SRC
0 1 2 3 4 5          0 1 2        0 1 2
^                    ^            ^
lsb                  lsb          lsb
```

A P E N D I X ------- F

BUS CONTROLLER INPUT

```
x1xx0xx00 10001010          1
xxxx0xx10 11000100          2
0xxx0xx11 11000100          3
xxxx0xx01 01000000          4
^10xxxxxx 00100000          5
 Jxxxx1xx 00100000          6
x1x1xxxxx 00010000          7
x0xxx1xxx 00010000          8
xxxxxx010 00000010          9
0xxxxx011 00000010         10
xxxxxx110 00000001         11
0xxxxx111 00000001         12
```

```
        INPUTS:
L                <-                    R
RDY,CYST,BRW,BCD,RES,C/D,R/W,S1,S0
```

```
        OUTPUTS:
L           <-             R
S0,S1,R/W,C/D,AS,DS,EN,LD
```